Load The Dataset (Week 2)

In [1]:

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

#ingest data
df = pd.read_csv('https://raw.githubusercontent.com/Christine971224/Analytics-2023/maste
df.head()
```

Out[1]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

5 rows × 36 columns

In [2]:

```
#basic information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 36 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
```

```
 11  babies                          119390 non-null  int64  
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
 31  reservation_status_date         119390 non-null  object 
 32  name                            119390 non-null  object 
 33  email                           119390 non-null  object 
 34  phone-number                    119390 non-null  object 
 35  credit_card                     119390 non-null  object 
dtypes: float64(4), int64(16), object(16)
memory usage: 32.8+ MB
```

In [3]:

```
df.isnull().mean()
```

Out[3]:

```
hotel                             0.000000
is_canceled                       0.000000
lead_time                         0.000000
arrival_date_year                 0.000000
arrival_date_month                0.000000
arrival_date_week_number          0.000000
arrival_date_day_of_month         0.000000
stays_in_weekend_nights           0.000000
stays_in_week_nights              0.000000
adults                            0.000000
children                          0.000034
babies                            0.000000
meal                              0.000000
country                           0.004087
market_segment                    0.000000
distribution_channel              0.000000
is_repeated_guest                 0.000000
previous_cancellations            0.000000
previous_bookings_not_canceled    0.000000
reserved_room_type                0.000000
assigned_room_type                0.000000
booking_changes                   0.000000
deposit_type                      0.000000
agent                             0.136862
company                           0.943069
days_in_waiting_list              0.000000
customer_type                     0.000000
adr                               0.000000
required_car_parking_spaces       0.000000
```

```
total_of_special_requests        0.000000
reservation_status               0.000000
reservation_status_date          0.000000
name                             0.000000
email                            0.000000
phone-number                     0.000000
credit_card                      0.000000
dtype: float64
```

In [4]:

```python
# adults, babies and children can't be zero at same time, so dropping the rows having a
filter = (df.children == 0) & (df.adults == 0) & (df.babies == 0)
df[filter]
```

Out[4]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
2224	Resort Hotel	0	1	2015	October	41
2409	Resort Hotel	0	0	2015	October	42
3181	Resort Hotel	0	36	2015	November	47
3684	Resort Hotel	0	165	2015	December	53
3708	Resort Hotel	0	165	2015	December	53
...	...	...	...	...	...	...
115029	City Hotel	0	107	2017	June	26
115091	City Hotel	0	1	2017	June	26
116251	City Hotel	0	44	2017	July	28
116534	City Hotel	0	2	2017	July	28
117087	City Hotel	0	170	2017	July	30

180 rows × 36 columns

In [5]:

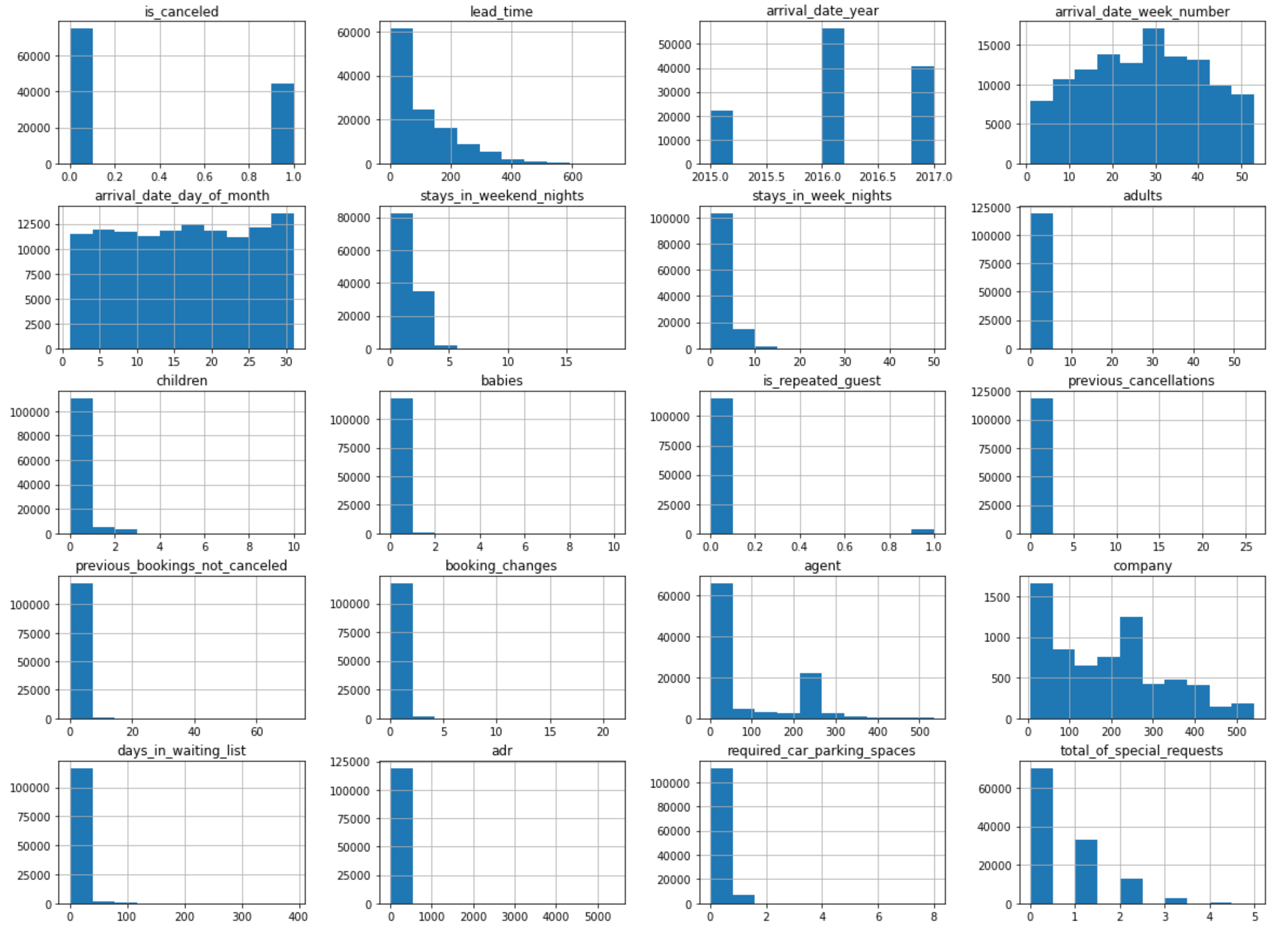
```
# transpose the resulting DataFrame
df.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.99]).T
```

Out[5]:

	count	mean	std	min	1%	5%	10%	2!
is_canceled	119390.0	0.370416	0.482918	0.00	0.0	0.0	0.0	0.
lead_time	119390.0	104.011416	106.863097	0.00	0.0	0.0	3.0	18.
arrival_date_year	119390.0	2016.156554	0.707476	2015.00	2015.0	2015.0	2015.0	2016.
arrival_date_week_number	119390.0	27.165173	13.605138	1.00	2.0	5.0	8.0	16.
arrival_date_day_of_month	119390.0	15.798241	8.780829	1.00	1.0	2.0	4.0	8.
stays_in_weekend_nights	119390.0	0.927599	0.998613	0.00	0.0	0.0	0.0	0.
stays_in_week_nights	119390.0	2.500302	1.908286	0.00	0.0	0.0	1.0	1.
adults	119390.0	1.856403	0.579261	0.00	1.0	1.0	1.0	2.
children	119386.0	0.103890	0.398561	0.00	0.0	0.0	0.0	0.
babies	119390.0	0.007949	0.097436	0.00	0.0	0.0	0.0	0.
is_repeated_guest	119390.0	0.031912	0.175767	0.00	0.0	0.0	0.0	0.
previous_cancellations	119390.0	0.087118	0.844336	0.00	0.0	0.0	0.0	0.
previous_bookings_not_canceled	119390.0	0.137097	1.497437	0.00	0.0	0.0	0.0	0.
booking_changes	119390.0	0.221124	0.652306	0.00	0.0	0.0	0.0	0.
agent	103050.0	86.693382	110.774548	1.00	1.0	1.0	6.0	9.
company	6797.0	189.266735	131.655015	6.00	16.0	40.0	40.0	62.
days_in_waiting_list	119390.0	2.321149	17.594721	0.00	0.0	0.0	0.0	0.
adr	119390.0	101.831122	50.535790	-6.38	0.0	38.4	50.0	69.
required_car_parking_spaces	119390.0	0.062518	0.245291	0.00	0.0	0.0	0.0	0.
total_of_special_requests	119390.0	0.571363	0.792798	0.00	0.0	0.0	0.0	0.

In [6]:

```
import matplotlib.pyplot as plt

# generate histograms for all the columns
df.hist(figsize=(20,15))
plt.show()
```



EDA（Week 3）

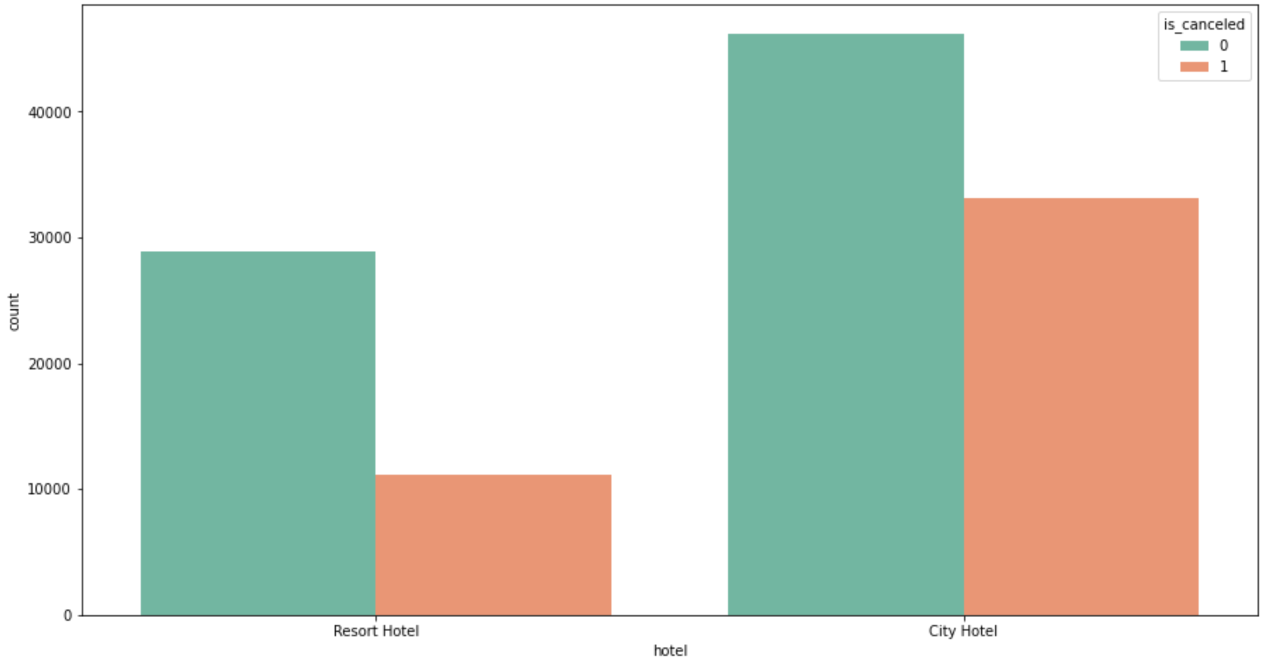
1. Hotel bookings and cancellations

In [7]:

```
# The number of hotel reservations and cancellations can directly show the actual numbe
import seaborn as sns
plt.figure(figsize=(15,8))
sns.countplot(x='hotel'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
```

Out[7]:

```
<AxesSubplot:xlabel='hotel', ylabel='count'>
```



In [8]:

```
#calculate the proportion of cancellations for each unique value in the 'hotel' column c
hotel_cancel=(df.loc[df['is_canceled']==1]['hotel'].value_counts()/df['hotel'].value_co
print('Hotel cancellations'.center(20),hotel_cancel,sep='\n')
```

```
Hotel cancellations
City Hotel      0.417270
Resort Hotel    0.277634
Name: hotel, dtype: float64
```

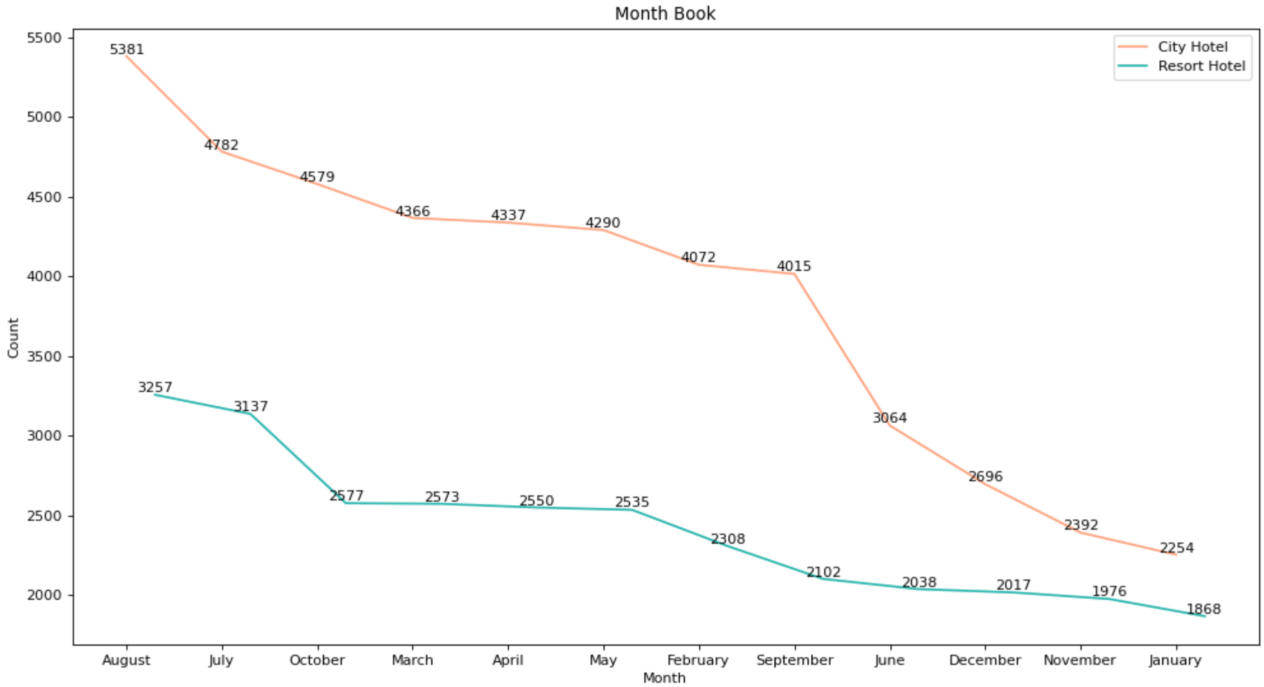
Comment: City Hotel's booking volume and cancellation volume are both higher than Resort Hotel's, but Resort Hotel's cancellation rate is 27.8%, while City Hotel's cancellation rate reaches 41.7%.

1. Hotel bookings by month

In [9]:

```
#create a plot to visualize the number of bookings for "City Hotel" and "Resort Hotel" c
city_hotel=df[(df['hotel']=='City Hotel') & (df['is_canceled']==0)]
resort_hotel=df[(df['hotel']=='Resort Hotel') & (df['is_canceled']==0)]
for i in [city_hotel,resort_hotel]:
    i.index=range(i.shape[0])

city_month=city_hotel['arrival_date_month'].value_counts()
resort_month=resort_hotel['arrival_date_month'].value_counts()
name=resort_month.index
x=list(range(len(city_month.index)))
y=city_month.values
x1=[i+0.3 for i in x]
y1=resort_month.values
width=0.3
plt.figure(figsize=(15,8),dpi=80)
plt.plot(x,y,label='City Hotel',color='lightsalmon')
plt.plot(x1,y1,label='Resort Hotel',color='lightseagreen')
plt.xticks(x,name)
plt.legend()
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Month Book')
```

```
for x,y in zip(x,y):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')

for x,y in zip(x1,y1):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')
```



Comment: Peak booking months are August and July. Preliminary judgment is that the long holiday caused the peak period.

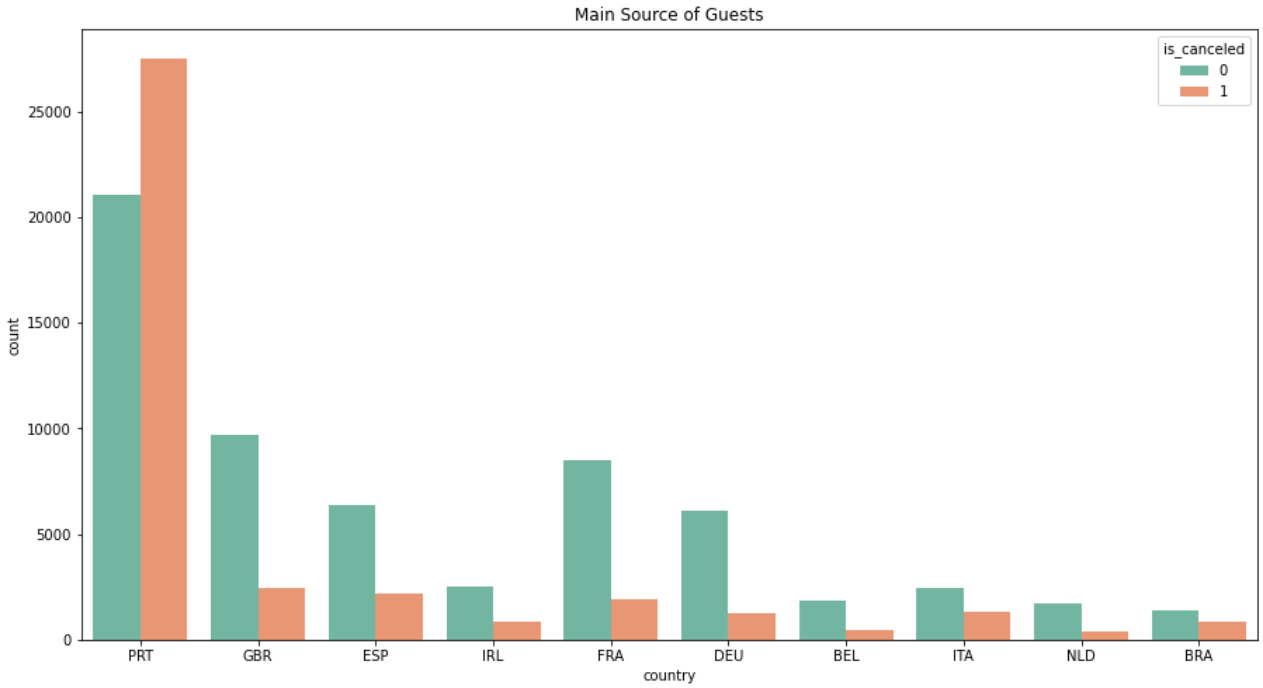
1. Customer origin and booking cancellation rate

In [10]:

```
#create a plot using Seaborn's countplot to visualize the top 10 countries from which b
country_book=df['country'].value_counts()[:10]
country_cancel=df[(df.country.isin (country_book.index)) & (df.is_canceled==1)]['countr
plt.figure(figsize=(15,8))
sns.countplot(x='country'
              ,data=df[df.country.isin (country_book.index)]
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Main Source of Guests')
```

Out[10]:

```
Text(0.5, 1.0, 'Main Source of Guests')
```



In [11]:

```
#calculate the cancellation rate for each of the top 10 countries (those with the highe
country_cancel_rate=(country_cancel/country_book).sort_values(ascending=False)
print('Customer cancellation rates by country'.center(10),country_cancel_rate,sep='\n')
```

```
Customer cancellation rates by country
PRT    0.566351
BRA    0.373201
ITA    0.353956
ESP    0.254085
IRL    0.246519
BEL    0.202391
GBR    0.202243
FRA    0.185694
NLD    0.183935
DEU    0.167147
Name: country, dtype: float64
```

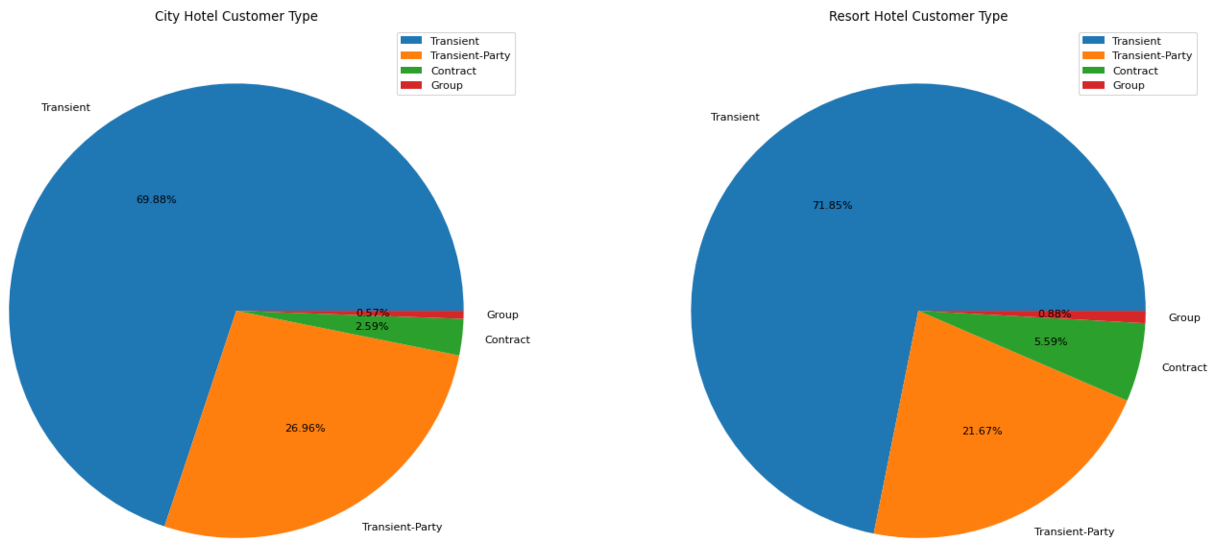
The peak season for both Resort hotel and City hotel is July and August in summer, and the main sources of tourists are European countries. This is in line with the characteristics of European tourists who prefer summer travel. It is necessary to focus on countries with high cancellation rates such as Portugal (PRT) and the United Kingdom (BRT). Main source of customers.

1. Customer type

In [12]:

```
#visualize the distribution of customer types for two types of hotels: City Hotel and Re
city_customer=city_hotel.customer_type.value_counts()
resort_customer=resort_hotel.customer_type.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_customer,labels=city_customer.index,autopct='%.2f%%')
plt.legend(loc=1)
plt.title('City Hotel Customer Type')
plt.subplot(1,2,2)
plt.pie(resort_customer,labels=resort_customer.index,autopct='%.2f%%')
```

```
plt.title('Resort Hotel Customer Type')
plt.legend()
plt.show()
```

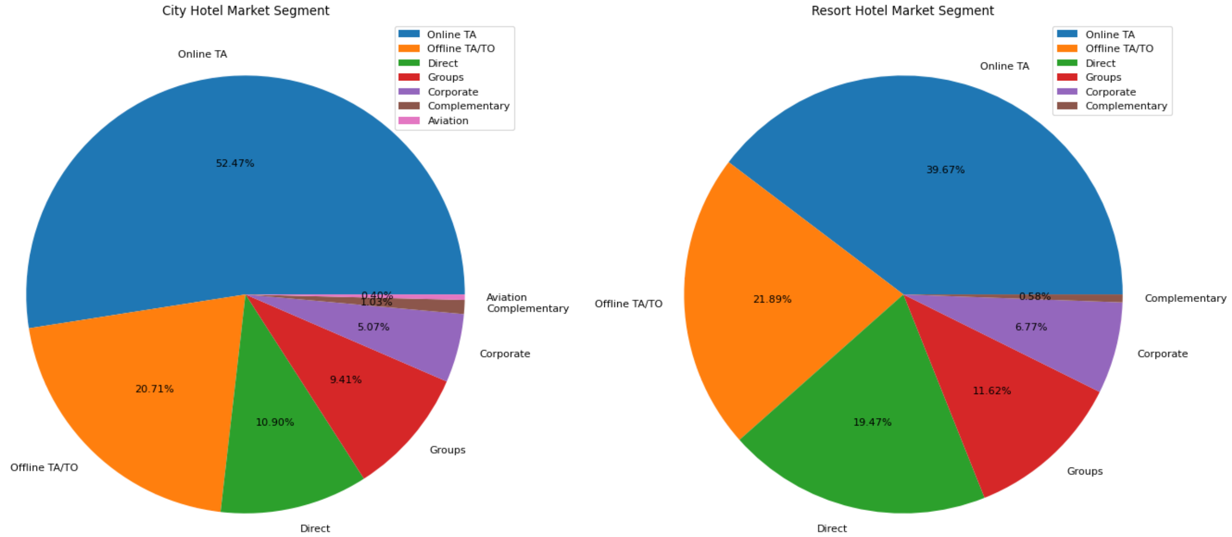


The main customer type of the hotel is transient travelers, accounting for about 70%.

1. Hotel booking method

In [13]:

```
#create pie charts to visualize the distribution of market segments for both City Hotel
city_segment=city_hotel.market_segment.value_counts()
resort_segment=resort_hotel.market_segment.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_segment,labels=city_segment.index,autopct='%.2f%%')
plt.legend()
plt.title('City Hotel Market Segment')
plt.subplot(1,2,2)
plt.pie(resort_segment,labels=resort_segment.index,autopct='%.2f%%')
plt.title('Resort Hotel Market Segment')
plt.legend()
plt.show()
```

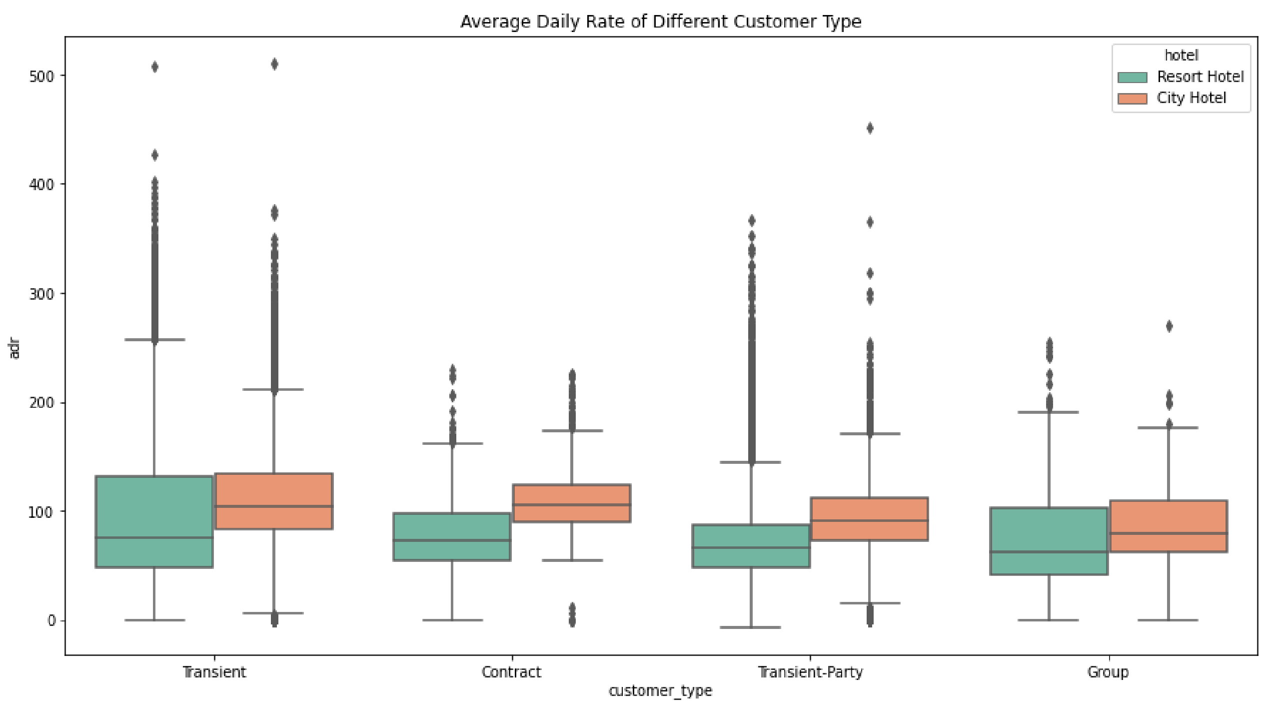


The customers of the two hotels mainly come from online travel agencies, which account for even more than 50% of the City Hotel; offline travel agencies come next, accounting for about 20%.

1. Average daily expenses of various types of passengers

In [14]:
```
#visualize the distribution of Average Daily Rate (adr) across different customer types,
plt.figure(figsize=(15,8))
sns.boxplot(x='customer_type'
            ,y='adr'
            ,hue='hotel'
            ,data=df[df.is_canceled==0]
            ,palette=sns.color_palette('Set2',2)
           )
plt.title('Average Daily Rate of Different Customer Type')
```

Out[14]:
```
Text(0.5, 1.0, 'Average Daily Rate of Different Customer Type')
```

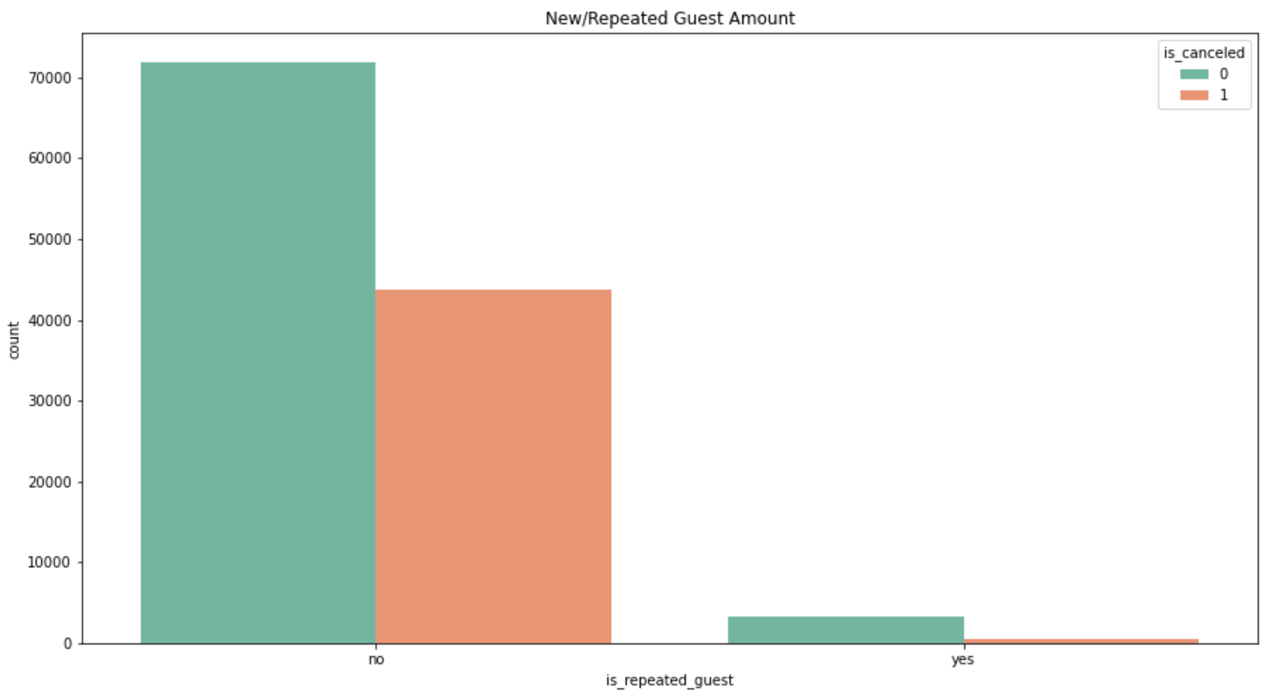


The average daily expenditure of all types of customers of City Hotel is higher than that of Resort Hotel; among the four types of customers, the consumption of individual travelers (Transient) is the highest and that of group travelers (Group) is the lowest.

7.Number of new and old customers and cancellation rate

In [15]:
```
# visualize the count of bookings, categorized by whether the guest is a repeated guest
plt.figure(figsize=(15,8))
sns.countplot(x='is_repeated_guest'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('New/Repeated Guest Amount')
plt.xticks(range(2),['no','yes'])
```

Out[15]:
```
([<matplotlib.axis.XTick at 0x1e129b89910>,
  <matplotlib.axis.XTick at 0x1e129b898e0>],
 [Text(0, 0, 'no'), Text(1, 0, 'yes')])
```



In [16]:
```
#calculate and printing the cancellation rates for new and repeated guests
guest_cancel=(df.loc[df['is_canceled']==1]['is_repeated_guest'].value_counts()/df['is_r
guest_cancel.index=['New Guest', 'Repeated Guest']
print('Cancellation rate for new and old customers'.center(15),guest_cancel,sep='\n')
```

```
Cancellation rate for new and old customers
New Guest         0.377851
Repeated Guest    0.144882
Name: is_repeated_guest, dtype: float64
```

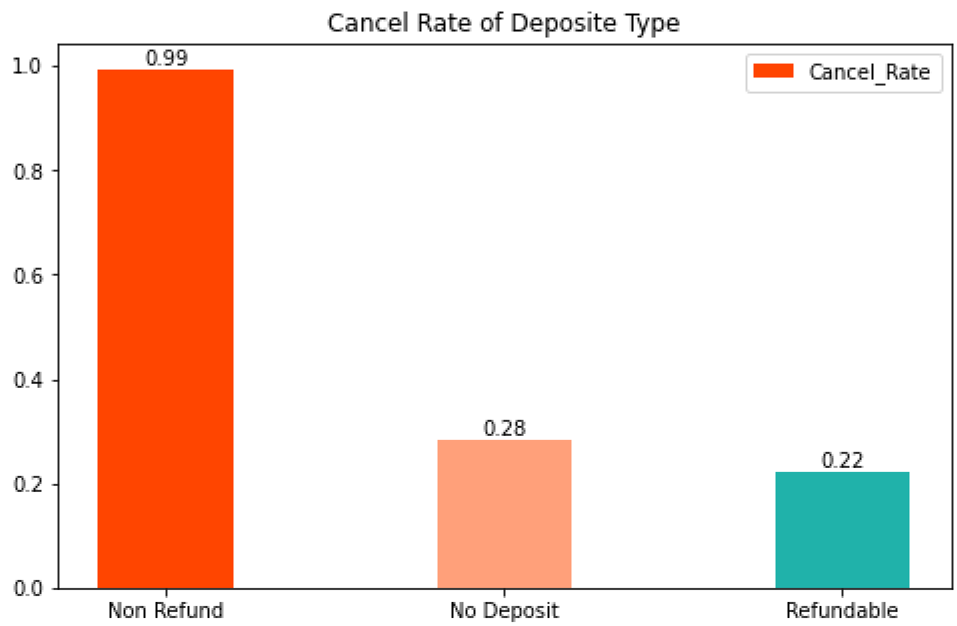
The cancellation rate for regular customers was 14.4%, while the cancellation rate for new customers reached 37.8%, which was 24 percentage points higher than that for regular customers.

1. Deposit method and reservation cancellation rate

In [17]:
```
print('Three deposit methods for booking quantity'.center(15),df['deposit_type'].value_
```

```
Three deposit methods for booking quantity
No Deposit    104641
Non Refund     14587
Refundable       162
Name: deposit_type, dtype: int64
```

In [18]:
```
#calculate the cancellation rates based on the 'deposit_type', and visualizing these ra
deposit_cancel=(df.loc[df['is_canceled']==1]['deposit_type'].value_counts()/df['deposit_
plt.figure(figsize=(8,5))
x=range(len(deposit_cancel.index))
y=deposit_cancel.values
plt.bar(x,y,label='Cancel_Rate',color=['orangered','lightsalmon','lightseagreen'],width
plt.xticks(x,deposit_cancel.index)
plt.legend()
plt.title('Cancel Rate of Deposite Type')
for x,y in zip(x,y):
    plt.text(x,y,'%.2f' % y,ha = 'center',va = 'bottom')
```

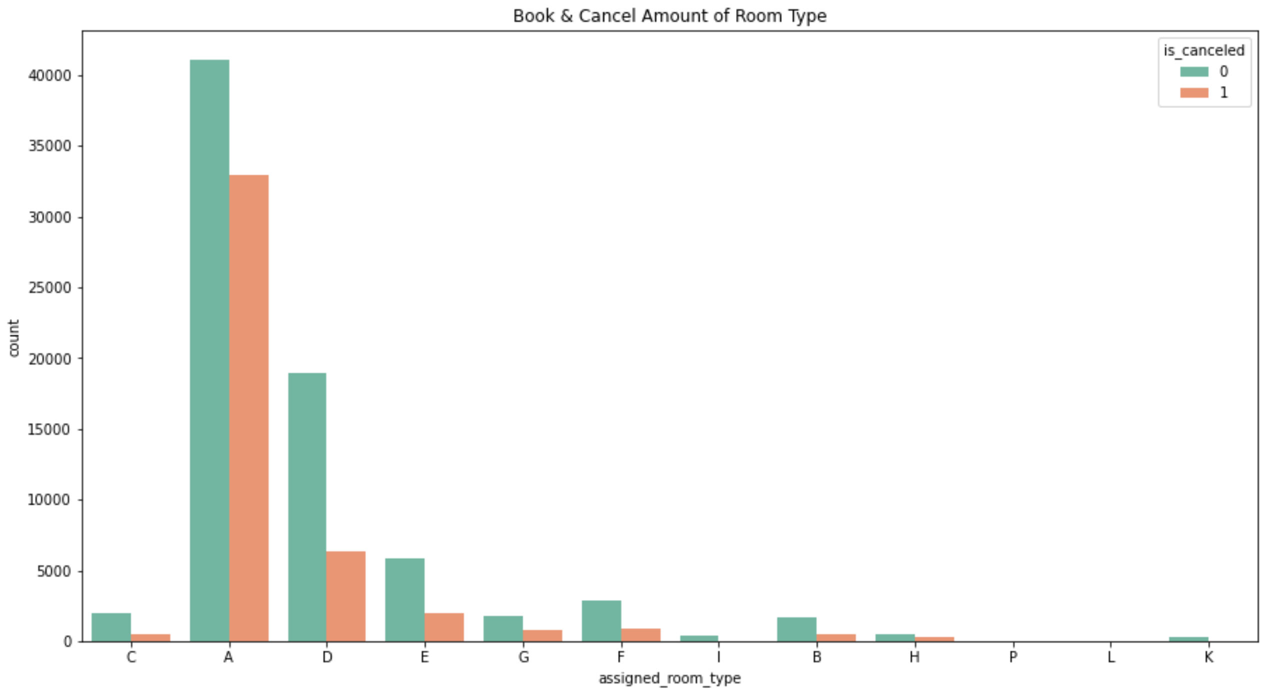


'No Deposit' is the method with the highest number of bookings and has a low cancellation rate, while the cancellation rate of non-refundable type is as high as 99%. This type of deposit method can be reduced to reduce Customer cancellation rate.

1. Room type and cancellation volume

In [19]:
```
#visualize the counts of bookings and cancellations across different assigned room type:
plt.figure(figsize=(15,8))
sns.countplot(x='assigned_room_type'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Book & Cancel Amount of Room Type')
```

Out[19]:
```
Text(0.5, 1.0, 'Book & Cancel Amount of Room Type')
```



In [20]:
```
#calculate cancellation rates for the top 7 assigned room types and printing them in de
room_cancel=df.loc[df['is_canceled']==1]['assigned_room_type'].value_counts()[:7]/df['a
print('Cancellation rates for different room types'.center(5),room_cancel.sort_values(a
```

```
Cancellation rates for different room types
A    0.444925
G    0.305523
E    0.252114
D    0.251244
F    0.247134
B    0.236708
C    0.187789
Name: assigned_room_type, dtype: float64
```

Among the top seven room types with the most bookings, the cancellation rates of room types A and G are higher than other room types, and the cancellation rate of room type A is as high as 44.5%.

Conclusion

1. The booking volume and cancellation rate of City Hotel are much higher than that of Resort Hotel. The hotel should conduct customer surveys to gain an in-depth understanding of the factors that cause customers to give up on bookings in order to reduce customer cancellation rates.

2. Hotels should make good use of the peak tourist season of July and August every year. They can increase prices appropriately while ensuring service quality to obtain more profits, and conduct preferential activities during the off-season (winter), such as Christmas sales and New Year activities, to reduce Hotel vacancy rate.

3. Hotels need to analyze customer profiles from major source countries such as Portugal and the United Kingdom, understand the attribute tags, preferences and consumption characteristics of these customers, and launch exclusive services to reduce customer cancellation rates.

4. Since individual travelers are the main customer group of hotels and have high consumption levels, hotels can increase the promotion and marketing of independent travelers through online and offline travel agencies, thereby attracting more tourists of this type.

5. The cancellation rate of new customers is 24% higher than that of old customers. Therefore, hotels should focus on the booking and check-in experience of new customers, and provide more guidance and benefits to new customers, such as providing discounts to first-time customers and conducting research on new customers. Provide feedback on satisfaction and dissatisfaction with your stay to improve future services and maintain good old customers.

6. The cancellation rate of non-refundable deposits is as high as 99%. Hotels should optimize this method, such as returning 50% of the deposit, or cancel this method directly to increase the occupancy rate.

7. The cancellation rate of room types A and G is much higher than that of other room types. The hotel should carefully confirm the room information with the customer when making a reservation, so that the customer can fully understand the room situation, avoid cognitive errors, and at the same time be able to understand the room facilities. Optimize and improve service levels.

Data Processing (Week 4)

In [21]:

```
#create a new DataFrame 'df1' from 'df'
df1=df.drop(labels=['reservation_status_date'],axis=1)
```

Handling Categorical Variables

In [22]:

```
cate=df1.columns[df1.dtypes == "object"].tolist() #getting the names of all columns in
#categorical variables expressed as numbers
num_cate=['agent','company','is_repeated_guest']
cate=cate+num_cate
```

In [23]:

```
import numpy as np #linear algebra
#creating a dictionary
results={}
for i in ['agent','company']:
    result=np.sort(df1[i].unique())
    results[i]=result
results
```

Out[23]:

```
{'agent': array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,  11.,
         12.,  13.,  14.,  15.,  16.,  17.,  19.,  20.,  21.,  22.,  23.,
         24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,  33.,  34.,
         35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  44.,  45.,  47.,
```

```
         50.,  52.,  53.,  54.,  55.,  56.,  57.,  58.,  59.,  60.,  61.,
         63.,  64.,  66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,
         75.,  77.,  78.,  79.,  81.,  82.,  83.,  85.,  86.,  87.,  88.,
         89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  98.,  99., 103.,
        104., 105., 106., 107., 110., 111., 112., 114., 115., 117., 118.,
        119., 121., 122., 126., 127., 128., 129., 132., 133., 134., 135.,
        138., 139., 141., 142., 143., 144., 146., 147., 148., 149., 150.,
        151., 152., 153., 154., 155., 156., 157., 158., 159., 162., 163.,
        165., 167., 168., 170., 171., 173., 174., 175., 177., 179., 180.,
        181., 182., 183., 184., 185., 187., 191., 192., 193., 195., 196.,
        197., 201., 205., 208., 210., 211., 213., 214., 215., 216., 219.,
        220., 223., 227., 229., 232., 234., 235., 236., 240., 241., 242.,
        243., 244., 245., 247., 248., 249., 250., 251., 252., 253., 254.,
        256., 257., 258., 261., 262., 265., 267., 269., 270., 273., 275.,
        276., 278., 280., 281., 282., 283., 285., 286., 287., 288., 289.,
        290., 291., 294., 295., 296., 298., 299., 300., 301., 302., 303.,
        304., 305., 306., 307., 308., 310., 313., 314., 315., 321., 323.,
        324., 325., 326., 327., 328., 330., 331., 332., 333., 334., 335.,
        336., 337., 339., 341., 344., 346., 348., 350., 352., 354., 355.,
        358., 359., 360., 363., 364., 367., 368., 370., 371., 375., 378.,
        384., 385., 387., 388., 390., 391., 393., 394., 397., 403., 404.,
        405., 406., 408., 410., 411., 414., 416., 418., 420., 423., 425.,
        426., 427., 429., 430., 431., 432., 433., 434., 436., 438., 440.,
        441., 444., 446., 449., 450., 451., 453., 454., 455., 459., 461.,
        464., 467., 468., 469., 472., 474., 475., 476., 479., 480., 481.,
        483., 484., 492., 493., 495., 497., 502., 508., 509., 510., 526.,
        527., 531., 535.,  nan]),
 'company': array([  6.,   8.,   9.,  10.,  11.,  12.,  14.,  16.,  18.,  20.,  22.,
         28.,  29.,  31.,  32.,  34.,  35.,  37.,  38.,  39.,  40.,  42.,
         43.,  45.,  46.,  47.,  48.,  49.,  51.,  52.,  53.,  54.,  59.,
         61.,  62.,  64.,  65.,  67.,  68.,  71.,  72.,  73.,  76.,  77.,
         78.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  88.,  91.,  92.,
         93.,  94.,  96.,  99., 100., 101., 102., 103., 104., 105., 106.,
        107., 108., 109., 110., 112., 113., 115., 116., 118., 120., 122.,
        126., 127., 130., 132., 135., 137., 139., 140., 142., 143., 144.,
        146., 148., 149., 150., 153., 154., 158., 159., 160., 163., 165.,
        167., 168., 169., 174., 178., 179., 180., 183., 184., 185., 186.,
        192., 193., 195., 197., 200., 202., 203., 204., 207., 209., 210.,
        212., 213., 215., 216., 217., 218., 219., 220., 221., 222., 223.,
        224., 225., 227., 229., 230., 232., 233., 234., 237., 238., 240.,
        242., 243., 245., 246., 250., 251., 253., 254., 255., 257., 258.,
        259., 260., 263., 264., 268., 269., 270., 271., 272., 273., 274.,
        275., 277., 278., 279., 280., 281., 282., 284., 286., 287., 288.,
        289., 290., 291., 292., 293., 297., 301., 302., 304., 305., 307.,
        308., 309., 311., 312., 313., 314., 316., 317., 318., 319., 320.,
        321., 323., 324., 325., 329., 330., 331., 332., 333., 334., 337.,
        338., 341., 342., 343., 346., 347., 348., 349., 350., 351., 352.,
        353., 355., 356., 357., 358., 360., 361., 362., 364., 365., 366.,
        367., 368., 369., 370., 371., 372., 373., 376., 377., 378., 379.,
        380., 382., 383., 384., 385., 386., 388., 390., 391., 392., 393.,
        394., 395., 396., 397., 398., 399., 400., 401., 402., 403., 405.,
        407., 408., 409., 410., 411., 412., 413., 415., 416., 417., 418.,
        419., 420., 421., 422., 423., 424., 425., 426., 428., 429., 433.,
        435., 436., 437., 439., 442., 443., 444., 445., 446., 447., 448.,
        450., 451., 452., 454., 455., 456., 457., 458., 459., 460., 461.,
        465., 466., 470., 477., 478., 479., 481., 482., 483., 484., 485.,
        486., 487., 489., 490., 491., 492., 494., 496., 497., 498., 499.,
        501., 504., 506., 507., 511., 512., 513., 514., 515., 516., 518.,
        520., 521., 523., 525., 528., 530., 531., 534., 539., 541., 543.,
         nan])}
```

In [24]:
```
# the agent and company columns have a large number of empty values and no 0 values, so
df1[['agent','company']]=df1[['agent','company']].fillna(0,axis=0)
```

In [25]:
```
df1.loc[:,cate].isnull().mean()
```

Out[25]:
```
hotel                   0.000000
arrival_date_month      0.000000
meal                    0.000000
country                 0.004087
market_segment          0.000000
distribution_channel    0.000000
reserved_room_type      0.000000
assigned_room_type      0.000000
deposit_type            0.000000
customer_type           0.000000
reservation_status      0.000000
name                    0.000000
email                   0.000000
phone-number            0.000000
credit_card             0.000000
agent                   0.000000
company                 0.000000
is_repeated_guest       0.000000
dtype: float64
```

In [26]:
```
#create new variables in_company and in_agent to classify passengers. If company and age
df1.loc[df1['company'] == 0,'in_company']='NO'
df1.loc[df1['company'] != 0,'in_company']='YES'
df1.loc[df1['agent'] == 0,'in_agent']='NO'
df1.loc[df1['agent'] != 0,'in_agent']='YES'
```

In [27]:
```
#create a new feature same_assignment. If the booked room type is consistent with the a
df1.loc[df1['reserved_room_type'] == df1['assigned_room_type'],'same_assignment']='Yes'
df1.loc[df1['reserved_room_type'] != df1['assigned_room_type'],'same_assignment']='No'
```

In [28]:
```
#delete four features except 'reserved_room_type', 'assigned_room_type', 'agent', 'comp
df1=df1.drop(labels=['reserved_room_type','assigned_room_type','agent','company'],axis=
```

In [29]:
```
#reset 'is_repeated_guest', frequent guests are marked as YES, non-repeated guests are 
df1['is_repeated_guest'][df1['is_repeated_guest']==0]='NO'
df1['is_repeated_guest'][df1['is_repeated_guest']==1]='YES'
```

In [30]:
```
#filling the missing values in the 'country' column of the DataFrame 'df1' with the mod
df1['country']=df1['country'].fillna(df1['country'].mode()[0])
```

In [31]:
```
for i in ['in_company','in_agent','same_assignment']:
    cate.append(i)

for i in ['reserved_room_type','assigned_room_type','agent','company']:
```

```
    cate.remove(i)
cate
```

Out[31]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'country',
 'market_segment',
 'distribution_channel',
 'deposit_type',
 'customer_type',
 'reservation_status',
 'name',
 'email',
 'phone-number',
 'credit_card',
 'is_repeated_guest',
 'in_company',
 'in_agent',
 'same_assignment']
```

In [32]:
```
#encoding categorical features
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe = oe.fit(df1.loc[:,cate])
df1.loc[:,cate] = oe.transform(df1.loc[:,cate])
```

Working With Continuous Variables

In [33]:
```
#to filter out continuous variables, you need to delete the label 'is_canceled' first.
col=df1.columns.tolist()
col.remove('is_canceled')
for i in cate:
    col.remove(i)
col
```

Out[33]:
```
['lead_time',
 'arrival_date_year',
 'arrival_date_week_number',
 'arrival_date_day_of_month',
 'stays_in_weekend_nights',
 'stays_in_week_nights',
 'adults',
 'children',
 'babies',
 'previous_cancellations',
 'previous_bookings_not_canceled',
 'booking_changes',
 'days_in_waiting_list',
 'adr',
 'required_car_parking_spaces',
 'total_of_special_requests']
```

In [34]:
```
df1[col].isnull().sum()
```

Out[34]:
```
lead_time                          0
arrival_date_year                  0
```

```
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          4
babies                            0
previous_cancellations            0
previous_bookings_not_canceled    0
booking_changes                   0
days_in_waiting_list              0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

In [35]:

```
#use mode to fill null values in xtrain children column
df1['children']=df1['children'].fillna(df1['children'].mode()[0])
```

In [36]:

```
#continuous variables are dimensionless
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss = ss.fit(df1.loc[:,col])
df1.loc[:,col] = ss.transform(df1.loc[:,col])
```

Correlation Coefficient of Each Variable

In [37]:

```
#calculating the correlation of all numerical columns with the 'is_canceled column' in
cor=df1.corr()
cor=abs(cor['is_canceled']).sort_values()
cor
```

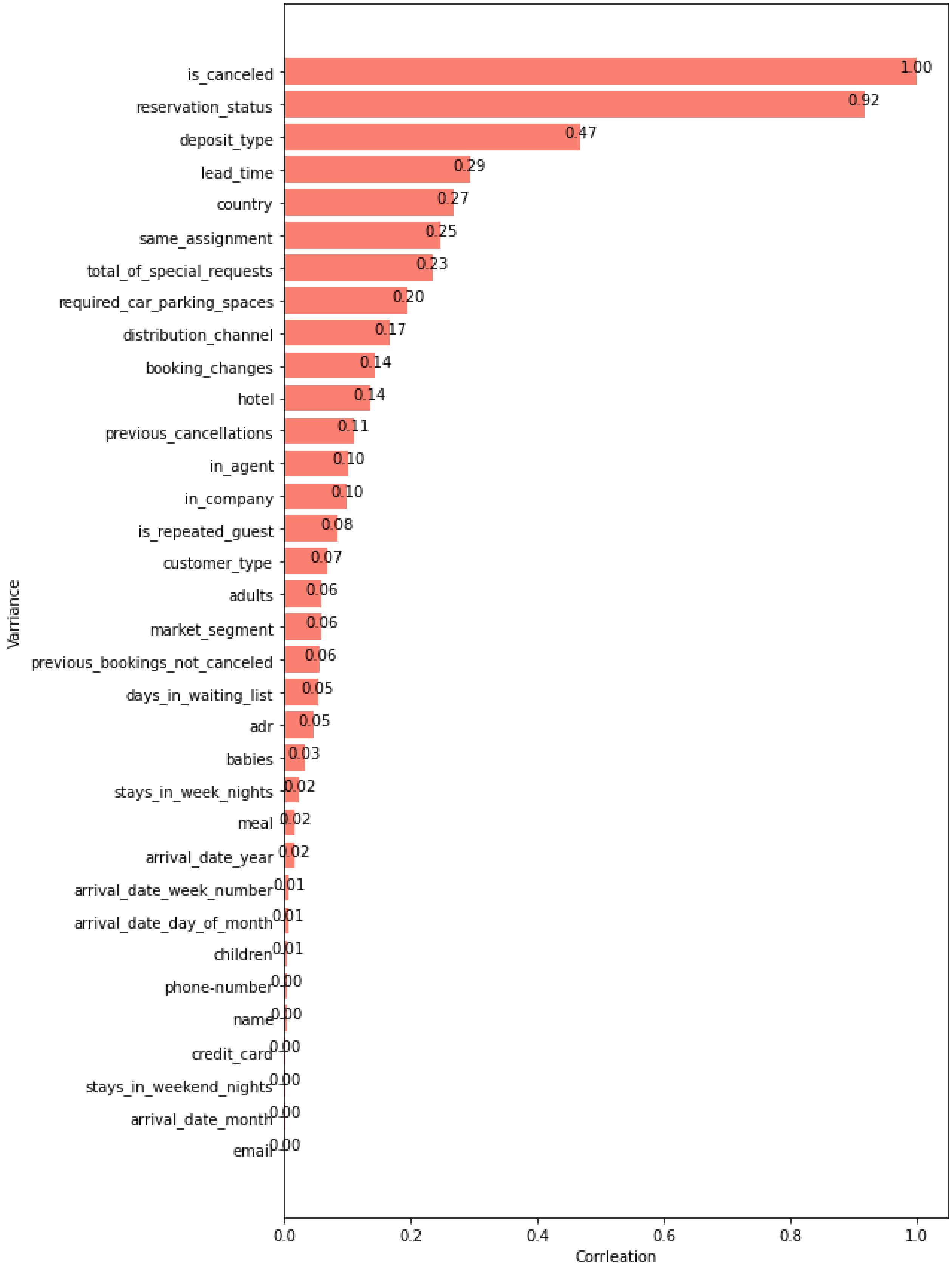
Out[37]:

```
email                             0.000723
arrival_date_month                0.001491
stays_in_weekend_nights           0.001791
credit_card                       0.002515
name                              0.004253
phone-number                      0.004342
children                          0.005036
arrival_date_day_of_month         0.006130
arrival_date_week_number          0.008148
arrival_date_year                 0.016660
meal                              0.017678
stays_in_week_nights              0.024765
babies                            0.032491
adr                               0.047557
days_in_waiting_list              0.054186
previous_bookings_not_canceled    0.057358
market_segment                    0.059338
adults                            0.060017
customer_type                     0.068140
is_repeated_guest                 0.084793
in_company                        0.099310
in_agent                          0.102068
previous_cancellations            0.110133
hotel                             0.136531
```

```
booking_changes                0.144381
distribution_channel           0.167600
required_car_parking_spaces    0.195498
total_of_special_requests      0.234658
same_assignment                0.247770
country                        0.267502
lead_time                      0.293123
deposit_type                   0.468634
reservation_status             0.917196
is_canceled                    1.000000
Name: is_canceled, dtype: float64
```

In [38]:

```
#create a horizontal bar plot using Matplotlib to visualize the absolute correlation va
plt.figure(figsize=(8,15))
x=range(len(cor.index))
name=cor.index
y=abs(cor.values)
plt.barh(x,y,color='salmon')
plt.yticks(x,name)
for x,y in zip(x,y):
    plt.text(y,x-0.1,'%.2f' % y,ha = 'center',va = 'bottom')
plt.xlabel('Corrleation')
plt.ylabel('Varriance')
plt.show()
```



The reservation status ('reservation_status') has the highest correlation with whether to cancel the reservation, reaching 0.92, but considering that it may cause the model to overfit in the future, it is deleted; the deposit type ('deposit_type') reaches 0.47, creating a characteristic Whether the reservation and assigned room type are consistent ('same_assignment') also has a correlation of 0.25.

In [39]:
```
#copy 'df1' with the column labeled 'reservation_status' dropped.
df2=df1.drop('reservation_status',axis=1)
```

Week 5

In [40]:
```
#dropping columns that are not useful
useless_col = ['email', 'phone-number', 'credit_card', 'name', 'days_in_waiting_list',
               'reservation_status', 'country', 'days_in_waiting_list']

df.drop(useless_col, axis = 1, inplace = True)
```

In [41]:
```
df.head()
```

Out[41]:

	hotel	is_canceled	lead_time	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mont
0	Resort Hotel	0	342	July	27	
1	Resort Hotel	0	737	July	27	
2	Resort Hotel	0	7	July	27	
3	Resort Hotel	0	13	July	27	
4	Resort Hotel	0	14	July	27	

5 rows × 26 columns

In [42]:
```
# creating numerical and categorical dataframes
cat_cols = [col for col in df.columns if df[col].dtype == 'O']
cat_cols
```

Out[42]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'market_segment',
 'distribution_channel',
 'reserved_room_type',
 'deposit_type',
 'customer_type',
 'reservation_status_date']
```

In [43]:
```
cat_df = df[cat_cols]
cat_df.head()
```

Out[43]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	deposit_
0	Resort Hotel	July	BB	Direct	Direct	C	No De
1	Resort Hotel	July	BB	Direct	Direct	C	No De
2	Resort Hotel	July	BB	Direct	Direct	A	No De
3	Resort Hotel	July	BB	Corporate	Corporate	A	No De
4	Resort Hotel	July	BB	Online TA	TA/TO	A	No De

In [44]:

```
#Convert 'reservation_status_date' to DateTime type
cat_df['reservation_status_date'] = pd.to_datetime(cat_df['reservation_status_date'])
#Extract the Year from the 'reservation_status_date'
cat_df['year'] = cat_df['reservation_status_date'].dt.year
#Extract the Month from the 'reservation_status_date'
cat_df['month'] = cat_df['reservation_status_date'].dt.month
#Extract the Day from the 'reservation_status_date'
cat_df['day'] = cat_df['reservation_status_date'].dt.day
```

In [45]:

```
cat_df.drop(['reservation_status_date','arrival_date_month'] , axis = 1, inplace = True
```

In [46]:

```
cat_df.head(15)
```

Out[46]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
0	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
1	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
2	Resort Hotel	BB	Direct	Direct	A	No Deposit	Transier
3	Resort Hotel	BB	Corporate	Corporate	A	No Deposit	Transier
4	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
5	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
6	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
7	Resort Hotel	FB	Direct	Direct	C	No Deposit	Transier

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
8	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
9	Resort Hotel	HB	Offline TA/TO	TA/TO	D	No Deposit	Transier
10	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier
11	Resort Hotel	HB	Online TA	TA/TO	D	No Deposit	Transier
12	Resort Hotel	BB	Online TA	TA/TO	D	No Deposit	Transier
13	Resort Hotel	HB	Online TA	TA/TO	G	No Deposit	Transier
14	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier

In [47]:

```
# printing unique values of each column
for col in cat_df.columns:
    print(f"{col}: \n{cat_df[col].unique()}\n")
```

```
hotel:
['Resort Hotel' 'City Hotel']

meal:
['BB' 'FB' 'HB' 'SC' 'Undefined']

market_segment:
['Direct' 'Corporate' 'Online TA' 'Offline TA/TO' 'Complementary' 'Groups'
 'Undefined' 'Aviation']

distribution_channel:
['Direct' 'Corporate' 'TA/TO' 'Undefined' 'GDS']

reserved_room_type:
['C' 'A' 'D' 'E' 'G' 'F' 'H' 'L' 'P' 'B']

deposit_type:
['No Deposit' 'Refundable' 'Non Refund']

customer_type:
['Transient' 'Contract' 'Transient-Party' 'Group']

year:
[2015 2014 2016 2017]

month:
[ 7  5  4  6  3  8  9  1 11 10 12  2]

day:
[ 1  2  3  6 22 23  5  7  8 11 15 16 29 19 18  9 13  4 12 26 17 10 20 14
 30 28 25 21 27 24 31]
```

In [48]:

```
# encoding categorical variables, which can be in text/string format, into numerical fo

cat_df['hotel'] = cat_df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

cat_df['meal'] = cat_df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined': 

cat_df['market_segment'] = cat_df['market_segment'].map({'Direct': 0, 'Corporate': 1, '
                                                          'Complementary': 4, 'Groups'

cat_df['distribution_channel'] = cat_df['distribution_channel'].map({'Direct': 0, 'Corp
                                                                      'GDS': 4})

cat_df['reserved_room_type'] = cat_df['reserved_room_type'].map({'C': 0, 'A': 1, 'D': 2
                                                                  'L': 7, 'B': 8})

cat_df['deposit_type'] = cat_df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1, 

cat_df['customer_type'] = cat_df['customer_type'].map({'Transient': 0, 'Contract': 1, '

cat_df['year'] = cat_df['year'].map({2015: 0, 2014: 1, 2016: 2, 2017: 3})
```

In [49]:

```
cat_df.head(15)
```

Out[49]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_type
0	0	0	0	0	0.0	0	(
1	0	0	0	0	0.0	0	(
2	0	0	0	0	1.0	0	(
3	0	0	1	1	1.0	0	(
4	0	0	2	2	1.0	0	(
5	0	0	2	2	1.0	0	(
6	0	0	0	0	0.0	0	(
7	0	1	0	0	0.0	0	(
8	0	0	2	2	1.0	0	(
9	0	2	3	2	2.0	0	(
10	0	0	2	2	3.0	0	(
11	0	2	2	2	2.0	0	(
12	0	0	2	2	2.0	0	(
13	0	2	2	2	4.0	0	(
14	0	0	2	2	3.0	0	(

In [50]:

```
num_df = df.drop(columns = cat_cols, axis = 1)
num_df.drop('is_canceled', axis = 1, inplace = True)
num_df
```

Out[50]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_i
0	342	27	1	0	
1	737	27	1	0	
2	7	27	1	0	
3	13	27	1	0	
4	14	27	1	0	
...	...	...	...	...	
119385	23	35	30	2	
119386	102	35	31	2	
119387	34	35	31	2	
119388	109	35	31	2	
119389	205	35	29	2	

119390 rows × 16 columns

In [51]:

```
num_df.var()
```

Out[51]:

```
lead_time                       11419.721511
arrival_date_week_number          185.099790
arrival_date_day_of_month          77.102966
stays_in_weekend_nights             0.997229
stays_in_week_nights                3.641554
adults                              0.335543
children                            0.158851
babies                              0.009494
is_repeated_guest                   0.030894
previous_cancellations              0.712904
previous_bookings_not_canceled      2.242317
agent                           12271.000405
company                         17333.042879
adr                              2553.866100
required_car_parking_spaces         0.060168
total_of_special_requests           0.628529
dtype: float64
```

In [52]:

```
# normalizing numerical variables, uses the natural logarithm to transform the data.
#It's essential to add 1 before taking the log to handle instances where the column valu
num_df['lead_time'] = np.log(num_df['lead_time'] + 1)
num_df['arrival_date_week_number'] = np.log(num_df['arrival_date_week_number'] + 1)
num_df['arrival_date_day_of_month'] = np.log(num_df['arrival_date_day_of_month'] + 1)
num_df['agent'] = np.log(num_df['agent'] + 1)
num_df['company'] = np.log(num_df['company'] + 1)
num_df['adr'] = np.log(num_df['adr'] + 1)
```

In [53]:

```
num_df.var()
```

Out[53]:

```
lead_time                         2.591420
arrival_date_week_number          0.441039
arrival_date_day_of_month         0.506267
stays_in_weekend_nights           0.997229
stays_in_week_nights              3.641554
adults                            0.335543
children                          0.158851
babies                            0.009494
is_repeated_guest                 0.030894
previous_cancellations            0.712904
previous_bookings_not_canceled    2.242317
agent                             2.536204
company                           0.755665
adr                               0.540353
required_car_parking_spaces       0.060168
total_of_special_requests         0.628529
dtype: float64
```

In [54]:

```
num_df['adr'] = num_df['adr'].fillna(value = num_df['adr'].mean())
num_df.head(15)
```

Out[54]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_we
0	5.837730	3.332205	0.693147	0	
1	6.603944	3.332205	0.693147	0	
2	2.079442	3.332205	0.693147	0	
3	2.639057	3.332205	0.693147	0	
4	2.708050	3.332205	0.693147	0	
5	2.708050	3.332205	0.693147	0	
6	0.000000	3.332205	0.693147	0	
7	2.302585	3.332205	0.693147	0	
8	4.454347	3.332205	0.693147	0	
9	4.330733	3.332205	0.693147	0	
10	3.178054	3.332205	0.693147	0	
11	3.583519	3.332205	0.693147	0	
12	4.234107	3.332205	0.693147	0	
13	2.944439	3.332205	0.693147	0	
14	3.637586	3.332205	0.693147	0	

Prepare the independent and dependent variables for a modeling task

In [55]:

```
#merging categorical and numerical dataframes
#X = pd.concat([cat_df, num_df], axis = 1)
#y = df['is_canceled']
x=df2.loc[:,df2.columns != 'is_canceled' ]
y=df2.loc[:,'is_canceled']
from sklearn.model_selection import train_test_split as tts
xtrain,xtest,ytrain,ytest=tts(x,y,test_size=0.3,random_state=90)
for i in [xtrain,xtest,ytrain,ytest]:
    i.index=range(i.shape[0])
```

In [56]:

```
x.shape, y.shape
```

Out[56]:

```
((119390, 32), (119390,))
```

In [57]:

```
xtrain.head()
```

Out[57]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	1.029252	1.192195	5.0	0.061361	-0.5
1	0.0	0.102829	-0.221286	8.0	-0.600156	-1.5

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
2	1.0	0.168334	1.192195	6.0	-0.085642	1.6
3	1.0	0.767233	1.192195	5.0	-0.012141	-0.8
4	0.0	-0.421208	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [58]:
```
xtest.head()
```

Out[58]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	-0.963961	-1.634768	2.0	1.898910	1.2
1	0.0	-0.861025	-0.221286	5.0	0.208365	0.3
2	0.0	1.431638	-0.221286	5.0	0.355369	1.7
3	0.0	-0.879741	-0.221286	11.0	0.722879	-1.2
4	0.0	-0.224694	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [59]:
```
ytrain.head(), ytest.head()
```

Out[59]:
```
(0    1
 1    0
 2    0
 3    1
 4    0
 Name: is_canceled, dtype: int64,
 0    0
 1    0
 2    0
 3    0
 4    0
 Name: is_canceled, dtype: int64)
```

Week 6 (2XGBoost)

In [60]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\zhumh\anaconda3\lib\site-packages (2.
0.1)
Requirement already satisfied: scipy in c:\users\zhumh\anaconda3\lib\site-packages (from
xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\zhumh\anaconda3\lib\site-packages (from
```

```
xgboost) (1.22.4)
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 23.0 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [61]:

```
import xgboost as xgb
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score as cvs,KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score as AUC
from sklearn.model_selection import cross_val_score
```

In [62]:

```
#Define the Model Variations
#base Model:This is the default parameters
model_1 = xgb.XGBClassifier(objective='binary:logistic', random_state=90)

#model 2: Adjust tree related hyperparameters
model_2 = xgb.XGBClassifier(objective='binary:logistic', max_depth=5, min_child_weight=

#model 3: Adjust boosting related hyperparameters
model_3 = xgb.XGBClassifier(objective='binary:logistic', learning_rate=0.01, n_estimato
```

In [63]:

```
#Train and Evaluate Each Model
models = [model_1, model_2, model_3]
model_names = ['Base Model', 'Tree Hyperparameters', 'Boosting Hyperparameters']
results = []

for i, model in enumerate(models):
    model.fit(xtrain, ytrain)

    # Predict
    train_pred = model.predict_proba(xtrain)[:,1]
    val_pred = model.predict_proba(xtest)[:,1]

    # Evaluate
    train_auc = roc_auc_score(ytrain, train_pred)
    val_auc = roc_auc_score(ytest, val_pred)

    results.append([model_names[i], train_auc, val_auc])

# Print results
print("Model Variation | Train AUC | Validation AUC")
print("----------------------------------------------")
for result in results:
    print(f"{result[0]:<25} | {result[1]:.4f}   | {result[2]:.4f}")
```

```
Model Variation | Train AUC | Validation AUC
----------------------------------------------
Base Model                | 0.9653   | 0.9429
Tree Hyperparameters      | 0.9550   | 0.9409
Boosting Hyperparameters  | 0.9302   | 0.9268
```

In [64]:

```
#Hyperparameter Tuning Using Optuna
import optuna
```

```
def objective(trial):
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-1)
    n_estimators = trial.suggest_int("n_estimators", 50, 500)
    max_depth = trial.suggest_int("max_depth", 1, 15)
    min_child_weight = trial.suggest_int("min_child_weight", 1, 7)
    subsample = trial.suggest_float("subsample", 0.5, 1.0)
    colsample_bytree = trial.suggest_float("colsample_bytree", 0.5, 1.0)

    model = xgb.XGBClassifier(
        objective='binary:logistic',
        learning_rate=learning_rate,
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_child_weight=min_child_weight,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        random_state=90
    )
    cv = 5
    return cross_val_score(model, xtrain, ytrain, n_jobs=-1, cv=cv).mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

best_params = study.best_params
print("Best parameters:", best_params)
```

```
[I 2023-11-04 21:22:04,558] A new study created in memory with name: no-name-9cdc74ff-c5
9d-4c3f-afac-541f523a3ef1
[I 2023-11-04 21:22:09,413] Trial 0 finished with value: 0.8693118788825795 and paramete
rs: {'learning_rate': 0.08812759585332756, 'n_estimators': 223, 'max_depth': 7, 'min_chi
ld_weight': 4, 'subsample': 0.741563721123603, 'colsample_bytree': 0.7814977010886848}.
Best is trial 0 with value: 0.8693118788825795.
[I 2023-11-04 21:22:12,343] Trial 1 finished with value: 0.871022965552584 and parameter
s: {'learning_rate': 0.09332913113799118, 'n_estimators': 51, 'max_depth': 15, 'min_chil
d_weight': 5, 'subsample': 0.534702827755787, 'colsample_bytree': 0.976805251713819}. Be
st is trial 1 with value: 0.871022965552584.
[I 2023-11-04 21:22:17,306] Trial 2 finished with value: 0.8676725926069633 and paramete
rs: {'learning_rate': 0.05963522210824153, 'n_estimators': 308, 'max_depth': 6, 'min_chi
ld_weight': 3, 'subsample': 0.9750399568848942, 'colsample_bytree': 0.7670371457894898}.
Best is trial 1 with value: 0.871022965552584.
[I 2023-11-04 21:22:24,494] Trial 3 finished with value: 0.8714058451502966 and paramete
rs: {'learning_rate': 0.09122673195324338, 'n_estimators': 449, 'max_depth': 7, 'min_chi
ld_weight': 7, 'subsample': 0.9035846700832619, 'colsample_bytree': 0.5311726439436149}.
Best is trial 3 with value: 0.8714058451502966.
[I 2023-11-04 21:22:30,115] Trial 4 finished with value: 0.8714297929327912 and paramete
rs: {'learning_rate': 0.08247587941915462, 'n_estimators': 249, 'max_depth': 8, 'min_chi
ld_weight': 5, 'subsample': 0.8624530039696451, 'colsample_bytree': 0.9635890846491784}.
Best is trial 4 with value: 0.8714297929327912.
[I 2023-11-04 21:22:32,319] Trial 5 finished with value: 0.8308425646992633 and paramete
rs: {'learning_rate': 0.023735680584566148, 'n_estimators': 94, 'max_depth': 5, 'min_chi
ld_weight': 5, 'subsample': 0.8279801052916881, 'colsample_bytree': 0.5414365427423187}.
Best is trial 4 with value: 0.8714297929327912.
[I 2023-11-04 21:22:34,800] Trial 6 finished with value: 0.867445259769762 and parameter
s: {'learning_rate': 0.08250004026277204, 'n_estimators': 59, 'max_depth': 10, 'min_chil
d_weight': 1, 'subsample': 0.8129533638699638, 'colsample_bytree': 0.7169674800933676}.
Best is trial 4 with value: 0.8714297929327912.
[I 2023-11-04 21:22:40,412] Trial 7 finished with value: 0.868881120185231 and parameter
s: {'learning_rate': 0.054427430965458885, 'n_estimators': 234, 'max_depth': 8, 'min_chi
ld_weight': 3, 'subsample': 0.5864293137324618, 'colsample_bytree': 0.7230895088177102}.
```

```
Best is trial 4 with value: 0.8714297929327912.
[I 2023-11-04 21:22:48,955] Trial 8 finished with value: 0.8744091950264181 and paramete
rs: {'learning_rate': 0.07572457342685779, 'n_estimators': 237, 'max_depth': 13, 'min_ch
ild_weight': 4, 'subsample': 0.7331186364640534, 'colsample_bytree': 0.851335857737399
8}. Best is trial 8 with value: 0.8744091950264181.
[I 2023-11-04 21:22:50,937] Trial 9 finished with value: 0.760843815708169 and parameter
s: {'learning_rate': 0.006623062486029713, 'n_estimators': 119, 'max_depth': 3, 'min_chi
ld_weight': 1, 'subsample': 0.5474880456551865, 'colsample_bytree': 0.5493243498710186}.
Best is trial 8 with value: 0.8744091950264181.
[I 2023-11-04 21:23:02,614] Trial 10 finished with value: 0.8729135138348878 and paramet
ers: {'learning_rate': 0.06679914575467459, 'n_estimators': 362, 'max_depth': 13, 'min_c
hild_weight': 7, 'subsample': 0.6756363368613415, 'colsample_bytree': 0.889007724868411
6}. Best is trial 8 with value: 0.8744091950264181.
[I 2023-11-04 21:23:14,723] Trial 11 finished with value: 0.8727220726042615 and paramet
ers: {'learning_rate': 0.06546890711930116, 'n_estimators': 372, 'max_depth': 13, 'min_c
hild_weight': 7, 'subsample': 0.6943091661224979, 'colsample_bytree': 0.884464121591755
2}. Best is trial 8 with value: 0.8744091950264181.
[I 2023-11-04 21:23:27,519] Trial 12 finished with value: 0.87151353571949 and parameter
s: {'learning_rate': 0.0720971085611593, 'n_estimators': 355, 'max_depth': 12, 'min_chil
d_weight': 6, 'subsample': 0.6660732035961646, 'colsample_bytree': 0.8885415414379597}.
Best is trial 8 with value: 0.8744091950264181.
[I 2023-11-04 21:23:52,607] Trial 13 finished with value: 0.8745527943834247 and paramet
ers: {'learning_rate': 0.04438400774570889, 'n_estimators': 476, 'max_depth': 15, 'min_c
hild_weight': 3, 'subsample': 0.6391079961306947, 'colsample_bytree': 0.863254441947802
4}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:24:19,193] Trial 14 finished with value: 0.8743972705312306 and paramet
ers: {'learning_rate': 0.04615154400800532, 'n_estimators': 466, 'max_depth': 15, 'min_c
hild_weight': 3, 'subsample': 0.613427203170613, 'colsample_bytree': 0.826578927771149
3}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:24:25,961] Trial 15 finished with value: 0.8719203831444752 and paramet
ers: {'learning_rate': 0.042300556371698966, 'n_estimators': 170, 'max_depth': 11, 'min_
child_weight': 2, 'subsample': 0.741395201165168, 'colsample_bytree': 0.831737276963734
9}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:24:54,356] Trial 16 finished with value: 0.8741698933091641 and paramet
ers: {'learning_rate': 0.03707206930232677, 'n_estimators': 421, 'max_depth': 15, 'min_c
hild_weight': 4, 'subsample': 0.6284534650600146, 'colsample_bytree': 0.994791576445751
1}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:25:09,021] Trial 17 finished with value: 0.874313499109135 and paramete
rs: {'learning_rate': 0.0742440643256568, 'n_estimators': 294, 'max_depth': 10, 'min_chi
ld_weight': 2, 'subsample': 0.7506633908386677, 'colsample_bytree': 0.6745859367282991}.
Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:25:21,801] Trial 18 finished with value: 0.8724109676290797 and paramet
ers: {'learning_rate': 0.05408135138501958, 'n_estimators': 186, 'max_depth': 13, 'min_c
hild_weight': 2, 'subsample': 0.5094527528931871, 'colsample_bytree': 0.924244741480835
5}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:25:41,658] Trial 19 finished with value: 0.871896448963794 and paramete
rs: {'learning_rate': 0.033824638484540426, 'n_estimators': 498, 'max_depth': 10, 'min_c
hild_weight': 4, 'subsample': 0.5816639936443788, 'colsample_bytree': 0.840421701996068
4}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:26:00,505] Trial 20 finished with value: 0.8706520276312968 and paramet
ers: {'learning_rate': 0.09950114319865348, 'n_estimators': 318, 'max_depth': 14, 'min_c
hild_weight': 4, 'subsample': 0.6399989219122135, 'colsample_bytree': 0.933248280276674
4}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:26:32,990] Trial 21 finished with value: 0.8744690551761505 and paramet
ers: {'learning_rate': 0.046837753668766184, 'n_estimators': 489, 'max_depth': 15, 'min_
child_weight': 3, 'subsample': 0.607021450928292, 'colsample_bytree': 0.828444281174204
5}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:26:40,626] Trial 22 finished with value: 0.8277793360604015 and paramet
ers: {'learning_rate': 0.05008076587324628, 'n_estimators': 499, 'max_depth': 1, 'min_ch
ild_weight': 3, 'subsample': 0.5793596860593816, 'colsample_bytree': 0.80442579665342}.
```

```
Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:27:05,905] Trial 23 finished with value: 0.8745408025950541 and paramet
ers: {'learning_rate': 0.0606267130043202, 'n_estimators': 405, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.7037907754050055, 'colsample_bytree': 0.860733946931097
6}. Best is trial 13 with value: 0.8745527943834247.
[I 2023-11-04 21:27:33,116] Trial 24 finished with value: 0.8752108823385498 and paramet
ers: {'learning_rate': 0.06141539076239223, 'n_estimators': 412, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.6486762138298297, 'colsample_bytree': 0.799392219222372
6}. Best is trial 24 with value: 0.8752108823385498.
[I 2023-11-04 21:27:55,115] Trial 25 finished with value: 0.874588701023583 and paramete
rs: {'learning_rate': 0.060496416569094436, 'n_estimators': 405, 'max_depth': 12, 'min_c
hild_weight': 2, 'subsample': 0.6992641853421093, 'colsample_bytree': 0.788119602429135
1}. Best is trial 24 with value: 0.8752108823385498.
[I 2023-11-04 21:28:15,790] Trial 26 finished with value: 0.8721118086256331 and paramet
ers: {'learning_rate': 0.05992633254074211, 'n_estimators': 415, 'max_depth': 11, 'min_c
hild_weight': 2, 'subsample': 0.6544817624857476, 'colsample_bytree': 0.791626454995698
4}. Best is trial 24 with value: 0.8752108823385498.
[I 2023-11-04 21:28:44,369] Trial 27 finished with value: 0.8727579828238445 and paramet
ers: {'learning_rate': 0.06589133187020722, 'n_estimators': 449, 'max_depth': 12, 'min_c
hild_weight': 1, 'subsample': 0.6513844419279522, 'colsample_bytree': 0.751051600260904
2}. Best is trial 24 with value: 0.8752108823385498.
[I 2023-11-04 21:29:10,917] Trial 28 finished with value: 0.8757852690283018 and paramet
ers: {'learning_rate': 0.053731699258433435, 'n_estimators': 393, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.7029392979052219, 'colsample_bytree': 0.802836619251854
2}. Best is trial 28 with value: 0.8757852690283018.
[I 2023-11-04 21:29:32,524] Trial 29 finished with value: 0.8738228766826295 and paramet
ers: {'learning_rate': 0.08017688927954876, 'n_estimators': 389, 'max_depth': 12, 'min_c
hild_weight': 2, 'subsample': 0.7142216042678812, 'colsample_bytree': 0.806435229775859
4}. Best is trial 28 with value: 0.8757852690283018.
[I 2023-11-04 21:29:45,104] Trial 30 finished with value: 0.8737152190441424 and paramet
ers: {'learning_rate': 0.05440661993637384, 'n_estimators': 321, 'max_depth': 9, 'min_ch
ild_weight': 1, 'subsample': 0.7651747210418, 'colsample_bytree': 0.7685655304132991}. B
est is trial 28 with value: 0.8757852690283018.
[I 2023-11-04 21:30:14,435] Trial 31 finished with value: 0.8758929352574076 and paramet
ers: {'learning_rate': 0.041614716074993646, 'n_estimators': 434, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.6865839724337697, 'colsample_bytree': 0.79333774485350
2}. Best is trial 31 with value: 0.8758929352574076.
[I 2023-11-04 21:30:38,011] Trial 32 finished with value: 0.8742177874423833 and paramet
ers: {'learning_rate': 0.06863190545384897, 'n_estimators': 345, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.6815830430197136, 'colsample_bytree': 0.793271727035423
1}. Best is trial 31 with value: 0.8758929352574076.
[I 2023-11-04 21:31:10,184] Trial 33 finished with value: 0.8767544726968828 and paramet
ers: {'learning_rate': 0.05377007107316485, 'n_estimators': 447, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.7095596395301156, 'colsample_bytree': 0.728980188696570
6}. Best is trial 33 with value: 0.8767544726968828.
[I 2023-11-04 21:31:42,768] Trial 34 finished with value: 0.8764074804104354 and paramet
ers: {'learning_rate': 0.05281376170027214, 'n_estimators': 437, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.7719865304936286, 'colsample_bytree': 0.722848543562012
2}. Best is trial 33 with value: 0.8767544726968828.
[I 2023-11-04 21:32:16,594] Trial 35 finished with value: 0.8781065616902559 and paramet
ers: {'learning_rate': 0.037241261299893025, 'n_estimators': 449, 'max_depth': 14, 'min_
child_weight': 1, 'subsample': 0.7749947790923544, 'colsample_bytree': 0.734075328396508
9}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:32:37,923] Trial 36 finished with value: 0.8763715608843483 and paramet
ers: {'learning_rate': 0.0377798655701192, 'n_estimators': 453, 'max_depth': 11, 'min_ch
ild_weight': 1, 'subsample': 0.7787934019861656, 'colsample_bytree': 0.694194947516132
1}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:33:05,472] Trial 37 finished with value: 0.8771971809453912 and paramet
ers: {'learning_rate': 0.03391877741388483, 'n_estimators': 452, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.7770825719714044, 'colsample_bytree': 0.680939206829266
```

```
1}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:33:14,751] Trial 38 finished with value: 0.8590932637340465 and paramet
ers: {'learning_rate': 0.028913306898230955, 'n_estimators': 441, 'max_depth': 5, 'min_c
hild_weight': 1, 'subsample': 0.7886764314886805, 'colsample_bytree': 0.66841969796099
5}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:33:43,117] Trial 39 finished with value: 0.8780108686365123 and paramet
ers: {'learning_rate': 0.024042351784616754, 'n_estimators': 464, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.8126532952781828, 'colsample_bytree': 0.6424237066930
2}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:33:53,115] Trial 40 finished with value: 0.8653632452008596 and paramet
ers: {'learning_rate': 0.01800345640971456, 'n_estimators': 268, 'max_depth': 9, 'min_ch
ild_weight': 1, 'subsample': 0.8429221536989826, 'colsample_bytree': 0.632925371460238
8}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:34:24,492] Trial 41 finished with value: 0.877460456216997 and paramete
rs: {'learning_rate': 0.032449591066474485, 'n_estimators': 469, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.7896038762087643, 'colsample_bytree': 0.712829801039501
8}. Best is trial 35 with value: 0.8781065616902559.
[I 2023-11-04 21:34:57,140] Trial 42 finished with value: 0.8782142758836515 and paramet
ers: {'learning_rate': 0.02489309137159252, 'n_estimators': 470, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.8037594111681576, 'colsample_bytree': 0.744289130104848
7}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:35:27,602] Trial 43 finished with value: 0.8779390453338065 and paramet
ers: {'learning_rate': 0.027183408677278482, 'n_estimators': 472, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.8088962554701807, 'colsample_bytree': 0.74708654760337
9}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:35:51,120] Trial 44 finished with value: 0.8763356778683924 and paramet
ers: {'learning_rate': 0.02178928063979376, 'n_estimators': 466, 'max_depth': 12, 'min_c
hild_weight': 1, 'subsample': 0.8109814207660525, 'colsample_bytree': 0.737724570559566
3}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:36:06,456] Trial 45 finished with value: 0.8673136192704195 and paramet
ers: {'learning_rate': 0.025393844832578578, 'n_estimators': 479, 'max_depth': 7, 'min_c
hild_weight': 1, 'subsample': 0.8691962608300723, 'colsample_bytree': 0.758830662536795
7}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:36:30,745] Trial 46 finished with value: 0.8753904105281473 and paramet
ers: {'learning_rate': 0.014540683367996663, 'n_estimators': 472, 'max_depth': 13, 'min_
child_weight': 5, 'subsample': 0.8069595265511867, 'colsample_bytree': 0.703560736135705
2}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:36:56,088] Trial 47 finished with value: 0.8776638419875887 and paramet
ers: {'learning_rate': 0.02709030243617631, 'n_estimators': 379, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.8305320302940276, 'colsample_bytree': 0.747762429713390
9}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:37:14,033] Trial 48 finished with value: 0.8758929395527172 and paramet
ers: {'learning_rate': 0.02883972852339368, 'n_estimators': 367, 'max_depth': 11, 'min_c
hild_weight': 1, 'subsample': 0.8454997557760505, 'colsample_bytree': 0.74216163418131}.
Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:37:38,191] Trial 49 finished with value: 0.8743493942951345 and paramet
ers: {'learning_rate': 0.011387271214826989, 'n_estimators': 384, 'max_depth': 15, 'min_
child_weight': 6, 'subsample': 0.8854824678302325, 'colsample_bytree': 0.768165672265529
7}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:37:54,820] Trial 50 finished with value: 0.8757374135528686 and paramet
ers: {'learning_rate': 0.021324155705014808, 'n_estimators': 337, 'max_depth': 12, 'min_
child_weight': 1, 'subsample': 0.9114989922071147, 'colsample_bytree': 0.649388878753440
8}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:38:22,107] Trial 51 finished with value: 0.8781903238058476 and paramet
ers: {'learning_rate': 0.027302354890658975, 'n_estimators': 470, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.7977959746747388, 'colsample_bytree': 0.711545621215428
9}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:38:50,360] Trial 52 finished with value: 0.8777236756495788 and paramet
ers: {'learning_rate': 0.025999898171364798, 'n_estimators': 429, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.8271155382816331, 'colsample_bytree': 0.743204759704740
```

```
2}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:39:01,189] Trial 53 finished with value: 0.8598351445878152 and paramet
ers: {'learning_rate': 0.02287856110876713, 'n_estimators': 428, 'max_depth': 6, 'min_ch
ild_weight': 1, 'subsample': 0.8023379169024015, 'colsample_bytree': 0.703704497738138
8}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:39:23,929] Trial 54 finished with value: 0.874086145511271 and paramete
rs: {'learning_rate': 0.018006012333578337, 'n_estimators': 485, 'max_depth': 11, 'min_c
hild_weight': 2, 'subsample': 0.7530076116929616, 'colsample_bytree': 0.772743822250956
6}. Best is trial 42 with value: 0.8782142758836515.
[I 2023-11-04 21:40:00,218] Trial 55 finished with value: 0.8785492821088081 and paramet
ers: {'learning_rate': 0.027545658079945724, 'n_estimators': 459, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.8247933683832832, 'colsample_bytree': 0.744989288678358
5}. Best is trial 55 with value: 0.8785492821088081.
[I 2023-11-04 21:40:32,913] Trial 56 finished with value: 0.8779031658972754 and paramet
ers: {'learning_rate': 0.029249794079711497, 'n_estimators': 460, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.7377728822441506, 'colsample_bytree': 0.622396331415016
3}. Best is trial 55 with value: 0.8785492821088081.
[I 2023-11-04 21:41:13,826] Trial 57 finished with value: 0.8787287787994689 and paramet
ers: {'learning_rate': 0.03184794377311279, 'n_estimators': 483, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.798180679892407, 'colsample_bytree': 0.691043083798006}.
Best is trial 57 with value: 0.8787287787994689.
[I 2023-11-04 21:41:47,872] Trial 58 finished with value: 0.8772450607609119 and paramet
ers: {'learning_rate': 0.03817226179252208, 'n_estimators': 500, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.7607423387326648, 'colsample_bytree': 0.721646290019418
9}. Best is trial 57 with value: 0.8787287787994689.
[I 2023-11-04 21:42:06,544] Trial 59 finished with value: 0.8776758165947209 and paramet
ers: {'learning_rate': 0.03237259340951992, 'n_estimators': 210, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.7307502835691997, 'colsample_bytree': 0.697408314309328
8}. Best is trial 57 with value: 0.8787287787994689.
[I 2023-11-04 21:42:16,474] Trial 60 finished with value: 0.8754861458191014 and paramet
ers: {'learning_rate': 0.04074891280006036, 'n_estimators': 117, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.7908305523873354, 'colsample_bytree': 0.681159147839780
6}. Best is trial 57 with value: 0.8787287787994689.
[I 2023-11-04 21:42:55,304] Trial 61 finished with value: 0.8787646832919724 and paramet
ers: {'learning_rate': 0.02413502608919184, 'n_estimators': 476, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.8236533265203767, 'colsample_bytree': 0.719039620184264
7}. Best is trial 61 with value: 0.8787646832919724.
[I 2023-11-04 21:43:37,730] Trial 62 finished with value: 0.8789681041409254 and paramet
ers: {'learning_rate': 0.023233891568610426, 'n_estimators': 496, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.82175614114678, 'colsample_bytree': 0.718039062380322
2}. Best is trial 62 with value: 0.8789681041409254.
[I 2023-11-04 21:44:11,806] Trial 63 finished with value: 0.8788364944246345 and paramet
ers: {'learning_rate': 0.03164343581815943, 'n_estimators': 485, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8338947572808825, 'colsample_bytree': 0.726844989720313
2}. Best is trial 62 with value: 0.8789681041409254.
[I 2023-11-04 21:44:44,854] Trial 64 finished with value: 0.878992023288023 and paramete
rs: {'learning_rate': 0.03050682564503828, 'n_estimators': 492, 'max_depth': 15, 'min_ch
ild_weight': 2, 'subsample': 0.8282687386444202, 'colsample_bytree': 0.716084446249315
1}. Best is trial 64 with value: 0.878992023288023.
[I 2023-11-04 21:45:16,646] Trial 65 finished with value: 0.8780228181876722 and paramet
ers: {'learning_rate': 0.03128782466964291, 'n_estimators': 488, 'max_depth': 15, 'min_c
hild_weight': 3, 'subsample': 0.8250188081592399, 'colsample_bytree': 0.762283800267768
7}. Best is trial 64 with value: 0.878992023288023.
[I 2023-11-04 21:45:52,049] Trial 66 finished with value: 0.8783698183488537 and paramet
ers: {'learning_rate': 0.03416075614714271, 'n_estimators': 490, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8491904875057447, 'colsample_bytree': 0.721572910874285
5}. Best is trial 64 with value: 0.878992023288023.
[I 2023-11-04 21:46:26,522] Trial 67 finished with value: 0.8783578372987572 and paramet
ers: {'learning_rate': 0.03464028107063887, 'n_estimators': 487, 'max_depth': 15, 'min_c
hild_weight': 3, 'subsample': 0.854384139419205, 'colsample_bytree': 0.711477495606798
```

```
6}. Best is trial 64 with value: 0.878992023288023.
[I 2023-11-04 21:47:02,481] Trial 68 finished with value: 0.8790996988236328 and paramet
ers: {'learning_rate': 0.03162350223979247, 'n_estimators': 494, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8716793624150124, 'colsample_bytree': 0.726204737501221
5}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:47:41,504] Trial 69 finished with value: 0.8784655321632601 and paramet
ers: {'learning_rate': 0.020147724667301036, 'n_estimators': 499, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8706707693621702, 'colsample_bytree': 0.688634246630142
3}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:48:11,846] Trial 70 finished with value: 0.8781783828453069 and paramet
ers: {'learning_rate': 0.030500086010910414, 'n_estimators': 403, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.8357565706872081, 'colsample_bytree': 0.664084842903566
5}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:48:56,510] Trial 71 finished with value: 0.8787168206576899 and paramet
ers: {'learning_rate': 0.019902436877803853, 'n_estimators': 492, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8722598065765982, 'colsample_bytree': 0.694828423933917
3}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:49:41,928] Trial 72 finished with value: 0.8783219549986863 and paramet
ers: {'learning_rate': 0.016882688191082296, 'n_estimators': 481, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8259873191857862, 'colsample_bytree': 0.695953477962246
6}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:50:21,095] Trial 73 finished with value: 0.8777715440109407 and paramet
ers: {'learning_rate': 0.02141383668445341, 'n_estimators': 499, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.8860062891730759, 'colsample_bytree': 0.77859959411009
2}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:50:30,500] Trial 74 finished with value: 0.7993130532917982 and paramet
ers: {'learning_rate': 0.013507137079708537, 'n_estimators': 454, 'max_depth': 2, 'min_c
hild_weight': 2, 'subsample': 0.8589637764599275, 'colsample_bytree': 0.731737756423831
7}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:51:12,512] Trial 75 finished with value: 0.8788245047839188 and paramet
ers: {'learning_rate': 0.030630940331775748, 'n_estimators': 486, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8401088249383293, 'colsample_bytree': 0.725395424842021
2}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:51:48,232] Trial 76 finished with value: 0.8782860619603413 and paramet
ers: {'learning_rate': 0.031191323657362945, 'n_estimators': 483, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.8396954010681827, 'colsample_bytree': 0.686974179966087
7}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:52:22,866] Trial 77 finished with value: 0.8789322025119615 and paramet
ers: {'learning_rate': 0.023477846363626716, 'n_estimators': 440, 'max_depth': 15, 'min_
child_weight': 3, 'subsample': 0.9082432205380505, 'colsample_bytree': 0.725697126513822
3}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:52:52,046] Trial 78 finished with value: 0.8781424647509896 and paramet
ers: {'learning_rate': 0.03585152341852791, 'n_estimators': 423, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.9207452208792329, 'colsample_bytree': 0.729793648582318
9}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:53:25,221] Trial 79 finished with value: 0.8778194059293384 and paramet
ers: {'learning_rate': 0.024103863490649496, 'n_estimators': 442, 'max_depth': 15, 'min_
child_weight': 3, 'subsample': 0.8175496256003839, 'colsample_bytree': 0.756717120759423
4}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:53:56,756] Trial 80 finished with value: 0.8780347892153797 and paramet
ers: {'learning_rate': 0.0397080817946, 'n_estimators': 478, 'max_depth': 14, 'min_child
_weight': 3, 'subsample': 0.9283409202320871, 'colsample_bytree': 0.7144513319594955}. B
est is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:54:36,000] Trial 81 finished with value: 0.8781185291385387 and paramet
ers: {'learning_rate': 0.043802970682490644, 'n_estimators': 494, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8670282216949907, 'colsample_bytree': 0.705741000207332
4}. Best is trial 68 with value: 0.8790996988236328.
[I 2023-11-04 21:55:13,205] Trial 82 finished with value: 0.8792911643943466 and paramet
ers: {'learning_rate': 0.023595811653034385, 'n_estimators': 457, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.852572048504639, 'colsample_bytree': 0.672735922683966
```

```
3}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:55:46,942] Trial 83 finished with value: 0.878489475650445 and paramete
rs: {'learning_rate': 0.029864687871852154, 'n_estimators': 461, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.8425902113565219, 'colsample_bytree': 0.673521648933590
8}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:56:18,983] Trial 84 finished with value: 0.8780467444936189 and paramet
ers: {'learning_rate': 0.023883769712216764, 'n_estimators': 442, 'max_depth': 15, 'min_
child_weight': 4, 'subsample': 0.8522034586111861, 'colsample_bytree': 0.724228189061021
5}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:56:53,545] Trial 85 finished with value: 0.8783698169170838 and paramet
ers: {'learning_rate': 0.03284199524283562, 'n_estimators': 475, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.8874326348873242, 'colsample_bytree': 0.660271165500189
4}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:57:30,467] Trial 86 finished with value: 0.8792313307323564 and paramet
ers: {'learning_rate': 0.02544091736842485, 'n_estimators': 458, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8563178459004535, 'colsample_bytree': 0.676639625910850
8}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:57:59,837] Trial 87 finished with value: 0.8773527398759464 and paramet
ers: {'learning_rate': 0.02544121310475783, 'n_estimators': 435, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.8990366031569823, 'colsample_bytree': 0.75552631838877
4}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:58:35,899] Trial 88 finished with value: 0.8790159460145451 and paramet
ers: {'learning_rate': 0.0352506272039359, 'n_estimators': 453, 'max_depth': 15, 'min_ch
ild_weight': 2, 'subsample': 0.8598696512775162, 'colsample_bytree': 0.678358590440189
3}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:59:03,621] Trial 89 finished with value: 0.8780108471599647 and paramet
ers: {'learning_rate': 0.03653666731235973, 'n_estimators': 414, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.852639230016428, 'colsample_bytree': 0.669664214488951
7}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 21:59:40,766] Trial 90 finished with value: 0.879063814375907 and paramete
rs: {'learning_rate': 0.029821955167734265, 'n_estimators': 453, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8602048946986081, 'colsample_bytree': 0.675746876843684
8}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 22:00:16,405] Trial 91 finished with value: 0.8788843341505995 and paramet
ers: {'learning_rate': 0.028575922606011453, 'n_estimators': 452, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8768178149162464, 'colsample_bytree': 0.679946012442354
9}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 22:00:51,369] Trial 92 finished with value: 0.8790877607266318 and paramet
ers: {'learning_rate': 0.026796057654821993, 'n_estimators': 451, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8606674600460987, 'colsample_bytree': 0.679330267450892
9}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 22:01:21,795] Trial 93 finished with value: 0.8787287924012824 and paramet
ers: {'learning_rate': 0.02811475868581064, 'n_estimators': 400, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.8786088312164115, 'colsample_bytree': 0.675347416964574
6}. Best is trial 82 with value: 0.8792911643943466.
[I 2023-11-04 22:01:59,446] Trial 94 finished with value: 0.8794945959815733 and paramet
ers: {'learning_rate': 0.022812442834764628, 'n_estimators': 423, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.8951589091989977, 'colsample_bytree': 0.657623122829734
7}. Best is trial 94 with value: 0.8794945959815733.
[I 2023-11-04 22:02:08,704] Trial 95 finished with value: 0.8502746317121057 and paramet
ers: {'learning_rate': 0.023053047527801206, 'n_estimators': 424, 'max_depth': 4, 'min_c
hild_weight': 2, 'subsample': 0.8990095471689691, 'colsample_bytree': 0.64896040372569
8}. Best is trial 94 with value: 0.8794945959815733.
[I 2023-11-04 22:02:38,225] Trial 96 finished with value: 0.878298023681545 and paramete
rs: {'learning_rate': 0.020932495315070902, 'n_estimators': 416, 'max_depth': 15, 'min_c
hild_weight': 4, 'subsample': 0.8658298790568789, 'colsample_bytree': 0.628263923973588
7}. Best is trial 94 with value: 0.8794945959815733.
[I 2023-11-04 22:03:10,620] Trial 97 finished with value: 0.8788125373356358 and paramet
ers: {'learning_rate': 0.026025080089390695, 'n_estimators': 443, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.8928272932034194, 'colsample_bytree': 0.664835670203349
```

```
4}. Best is trial 94 with value: 0.8794945959815733.
[I 2023-11-04 22:03:45,024] Trial 98 finished with value: 0.8778672614047718 and paramet
ers: {'learning_rate': 0.03528250393795529, 'n_estimators': 461, 'max_depth': 15, 'min_c
hild_weight': 3, 'subsample': 0.8605217302163098, 'colsample_bytree': 0.702613206309210
5}. Best is trial 94 with value: 0.8794945959815733.
[I 2023-11-04 22:03:52,067] Trial 99 finished with value: 0.8600984148482265 and paramet
ers: {'learning_rate': 0.01836458541783125, 'n_estimators': 74, 'max_depth': 14, 'min_ch
ild_weight': 2, 'subsample': 0.9031374000318588, 'colsample_bytree': 0.654651793172799
5}. Best is trial 94 with value: 0.8794945959815733.
Best parameters: {'learning_rate': 0.022812442834764628, 'n_estimators': 423, 'max_dept
h': 15, 'min_child_weight': 2, 'subsample': 0.8951589091989977, 'colsample_bytree': 0.65
76231228297347}
```

In [65]:

```
#Printing the best hyperparameters and their corresponding cross-validation score
best_score = study.best_value
print(f"Best cross-validation score: {best_score}")
```

```
Best cross-validation score: 0.8794945959815733
```

Week 7 (Logistic)

In [66]:

```
import optuna
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
```

In [67]:

```
# Base Logistic Regression Model
base_lr = LogisticRegression(random_state=90)
base_lr.fit(xtrain, ytrain)
base_pred = base_lr.predict_proba(xtest)[:, 1]  # Get probabilities for the positive cl
base_auc = roc_auc_score(ytest, base_pred)

# Variation 1: Different regularization strength
lr_v1 = LogisticRegression(C=0.1, random_state=90)
lr_v1.fit(xtrain, ytrain)
pred_v1 = lr_v1.predict_proba(xtest)[:, 1]
auc_v1 = roc_auc_score(ytest, pred_v1)

# Variation 2: Different penalty
lr_v2 = LogisticRegression(penalty='l1', solver='liblinear', random_state=90)
lr_v2.fit(xtrain, ytrain)
pred_v2 = lr_v2.predict_proba(xtest)[:, 1]
auc_v2 = roc_auc_score(ytest, pred_v2)

# Variation 3: Different solver and multi-class strategy
lr_v3 = LogisticRegression(solver='newton-cg', multi_class='multinomial', random_state=
lr_v3.fit(xtrain, ytrain)
pred_v3 = lr_v3.predict_proba(xtest)[:, 1]
auc_v3 = roc_auc_score(ytest, pred_v3)
```

In [68]:

```
# Print out the AUC for each model
print(f"Variation 1 AUC: {auc_v1}")
print(f"Variation 2 AUC: {auc_v2}")
print(f"Variation 3 AUC: {auc_v3}")
```

```
Variation 1 AUC: 0.6097526629711576
Variation 2 AUC: 0.8669425437781042
Variation 3 AUC: 0.8663045670205596
```

In [69]:

```
# Create a DataFrame to display results
results = pd.DataFrame({
    'Model': ['Variation 1', 'Variation 2', 'Variation 3'],
    'C': [0.1, 1.0, 1.0],
    'Penalty': ['l2', 'l1', 'l2'],
    'Solver': ['lbfgs', 'liblinear', 'newton-cg'],
    'AUC': [auc_v1, auc_v2, auc_v3]
})

print(results)
```

```
         Model    C Penalty     Solver       AUC
0  Variation 1  0.1      l2      lbfgs  0.609753
1  Variation 2  1.0      l1  liblinear  0.866943
2  Variation 3  1.0      l2  newton-cg  0.866305
```

In [70]:

```
# Select the best model
best_auc = results['AUC'].max()
best_model = results.loc[results['AUC'].idxmax(), 'Model']
print(f"The best model is {best_model} with an AUC of {best_auc}")
```

```
The best model is Variation 2 with an AUC of 0.8669425437781042
```

Week8 (Random Forest)

In [71]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

base_rfc = RandomForestClassifier(random_state=90)
base_rfc.fit(xtrain, ytrain)
base_pred = base_rfc.predict(xtest)
base_auc = roc_auc_score(ytest, base_pred)
```

In [72]:

```
# Variation 1: Small number of trees, unlimited depth
rfc1 = RandomForestClassifier(n_estimators=10, max_depth=None, random_state=90)
rfc1.fit(xtrain, ytrain)
pred1 = rfc1.predict(xtest)
auc1 = roc_auc_score(ytest, pred1)

# Variation 2: Moderate number of trees, depth of 10
rfc2 = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=90)
rfc2.fit(xtrain, ytrain)
pred2 = rfc2.predict(xtest)
auc2 = roc_auc_score(ytest, pred2)

# Variation 3: Large number of trees, depth of 3
rfc3 = RandomForestClassifier(n_estimators=500, max_depth=3, random_state=90)
rfc3.fit(xtrain, ytrain)
pred3 = rfc3.predict(xtest)
auc3 = roc_auc_score(ytest, pred3)
```

In [73]:
```
print(f"Variation 1 AUC: {auc1}")
print(f"Variation 2 AUC: {auc2}")
print(f"Variation 3 AUC: {auc3}")
```

```
Variation 1 AUC: 0.8317632978170258
Variation 2 AUC: 0.8141940904686819
Variation 3 AUC: 0.6835733164827276
```

In [74]:
```
import pandas as pd

data = {
    'Variation': ['Variation 1', 'Variation 2', 'Variation 3'],
    'n_estimators': [10, 100, 500], # default is 100
    'max_depth': [None, 10, 3],
    'AUC': [auc1, auc2, auc3]
}

df = pd.DataFrame(data)
print(df)
```

```
     Variation  n_estimators  max_depth       AUC
0  Variation 1            10        NaN  0.831763
1  Variation 2           100       10.0  0.814194
2  Variation 3           500        3.0  0.683573
```

In [75]:
```
best_auc = max([auc1, auc2, auc3])
best_model = ['Variation 1', 'Variation 2', 'Variation 3'][[auc1, auc2, auc3].index(bes
print(f"The best model is {best_model} with an AUC of {best_auc}")
```

```
The best model is Variation 1 with an AUC of 0.8317632978170258
```

In [76]:
```
# Calculate training AUCs
auc1_train = roc_auc_score(ytrain, rfc1.predict(xtrain))
auc2_train = roc_auc_score(ytrain, rfc2.predict(xtrain))
auc3_train = roc_auc_score(ytrain, rfc3.predict(xtrain))

# Add training AUCs to the dataframe
df['AUC_Train'] = [auc1_train, auc2_train, auc3_train]
print(df)
```

```
     Variation  n_estimators  max_depth       AUC  AUC_Train
0  Variation 1            10        NaN  0.831763   0.989448
1  Variation 2           100       10.0  0.814194   0.817013
2  Variation 3           500        3.0  0.683573   0.685281
```

Week 9

In [77]:
```
from sklearn.metrics import accuracy_score

# Logistic Regression Evaluations
lr_metrics = []
for lr in [lr_v1, lr_v2, lr_v3]:
    val_pred = lr.predict_proba(xtest)[:, 1]
    val_auc = AUC(ytest, val_pred)
    val_accuracy = accuracy_score(ytest, lr.predict(xtest))
```

```
    lr_metrics.append([val_auc, val_accuracy])

# XGBoost Evaluations
xgb_metrics = []
for model in [model_1, model_2, model_3]:
    val_pred = model.predict_proba(xtest)[:, 1]
    val_auc = AUC(ytest, val_pred)
    val_accuracy = accuracy_score(ytest, model.predict(xtest))
    xgb_metrics.append([val_auc, val_accuracy])

# Random Forest Evaluations
rf_metrics = []
for rf in [rfc1, rfc2, rfc3]:
    val_pred = rf.predict_proba(xtest)[:, 1]
    val_auc = AUC(ytest, val_pred)
    val_accuracy = accuracy_score(ytest, rf.predict(xtest))
    rf_metrics.append([val_auc, val_accuracy])
```

In [78]:

```
# Combine all metrics into one DataFrame for display
all_metrics = lr_metrics + xgb_metrics + rf_metrics
model_variations = [
    'LR Variation 1', 'LR Variation 2', 'LR Variation 3',
    'XGB Base','XGB Tree Hyperparameters', 'XGB Boosting Hyperparameters',
    'RF Variation 1', 'RF Variation 2', 'RF Variation 3'
]

df_eval = pd.DataFrame(all_metrics,
                       columns=['Validation AUC', 'Validation Accuracy'],
                       index=model_variations)

print(df_eval)

# Finding the best model based on Validation AUC
best_model_index = df_eval['Validation AUC'].idxmax()
best_model_metrics = df_eval.loc[best_model_index]

print(f"The best model is {best_model_index} with a Validation AUC of {best_model_metri
```

```
                              Validation AUC  Validation Accuracy
LR Variation 1                      0.609753             0.626965
LR Variation 2                      0.866943             0.796410
LR Variation 3                      0.866305             0.796270
XGB Base                            0.942924             0.869140
XGB Tree Hyperparameters            0.940895             0.866823
XGB Boosting Hyperparameters        0.926824             0.850518
RF Variation 1                      0.927008             0.858922
RF Variation 2                      0.924105             0.847531
RF Variation 3                      0.877473             0.767764
The best model is XGB Base with a Validation AUC of 0.9429
```

In [79]:

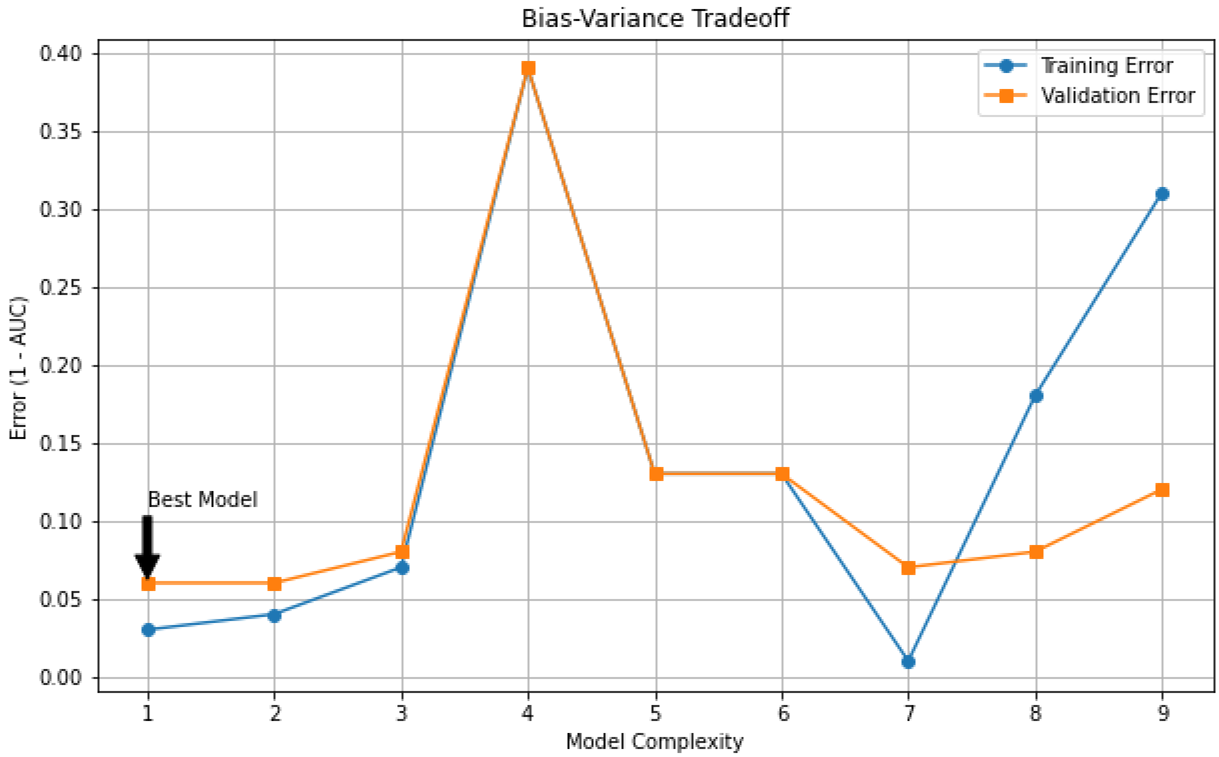
```
# Identify the best model from the models I trained
winning_model_name = best_model_index
winning_models = {
    'LR Base': base_lr, 'LR Variation 1': lr_v1, 'LR Variation 2': lr_v2, 'LR Variation
    'XGB Base': model_1, 'XGB Tree Hyperparameters': model_2, 'XGB Boosting Hyperparame
    'RF Base': base_rfc, 'RF Variation 1': rfc1, 'RF Variation 2': rfc2, 'RF Variation
}
```

```
# Select the best model
winning_model = winning_models[winning_model_name]

# Predict on the test set with the winning model
test_pred_proba = winning_model.predict_proba(xtest)[:, 1]
test_pred = winning_model.predict(xtest)

# Calculate AUC on the test set
test_auc = AUC(ytest, test_pred_proba)

# Calculate accuracy on the test set
test_accuracy = accuracy_score(ytest, test_pred)

# Print out the test metrics for the winning model
print(f"Winning Model: {winning_model_name}")
print(f"Test AUC: {test_auc:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Winning Model: XGB Base
Test AUC: 0.9429
Test Accuracy: 0.8691
```

In [80]:

```
import matplotlib.pyplot as plt

# Assuming you have these AUC scores for 9 models.
training_auc = [0.97, 0.96, 0.93, 0.61, 0.87, 0.87, 0.99, 0.82, 0.69]  # Replace with y
validation_auc = [0.94, 0.94, 0.92, 0.61, 0.87, 0.87, 0.93, 0.92, 0.88]  # Replace with

# Calculate the error as 1 - AUC.
training_error = [1 - auc for auc in training_auc]
validation_error = [1 - auc for auc in validation_auc]

# Assume model complexity increases with the model index.
model_complexity = list(range(1, 10))  # Models 1 to 9

# Create the plot.
plt.figure(figsize=(10, 6))
plt.plot(model_complexity, training_error, label='Training Error', marker='o')
plt.plot(model_complexity, validation_error, label='Validation Error', marker='s')

# Annotating the model with the lowest validation error
min_val_error_index = validation_error.index(min(validation_error))
plt.annotate('Best Model',
             xy=(model_complexity[min_val_error_index], validation_error[min_val_error_
             xytext=(model_complexity[min_val_error_index], validation_error[min_val_er
             arrowprops=dict(facecolor='black', shrink=0.05))

# Adding details to the plot.
plt.title('Bias-Variance Tradeoff')
plt.xlabel('Model Complexity')
plt.ylabel('Error (1 - AUC)')
plt.legend()
plt.grid(True)
plt.show()
```


Bias-Variance Tradeoff
Training Error
Validation Error
Best Model
Error (1 - AUC)
Model Complexity
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
8
9